Universidade Federal do Paraná

Everton Ataide
Sergio Henrique Costa
Renam Martinez de Oliveira

Ammit: Uma proposta de linguagem de descrição de casos de teste para exercícios de programação

Curitiba, 2017

Universidade Federal do Paraná
Everton Ataide
Sergio Henrique Costa
Renam Martinez de Oliveira

Ammit: Uma proposta de linguagem de descrição de casos de teste para exercícios de programação

Trabalho apresentado à disciplina de Trabalho de Conclusão de Curso, do curso de Tecnologia em Análise e Desenvolvimento de Sistemas, Setor de Educação Profissional e Tecnológica da Universidade Federal do Paraná.

Orientador: Prof. Dr. Alexander Robert Kutzke

Curitiba, 2017

RESUMO

Este trabalho é de uma proposta de melhoria para a forma como softwares de correção de código tratam a geração de casos de teste. Contando com a elaboração de uma linguagem interpretada própria com geração de conteúdo a partir de expressões regulares para a criação dos casos de teste, os autores desenvolveram uma aplicação web que fornece aos usuários um ambiente para cadastro de questões e envio de tentativas de resposta, para servir de prova de conceito para a linguagem em si.

É possibilitado durante a utilização do sistema que os usuários gerem casos de teste automaticamente através da linguagem Ammit ou manualmente caso suas necessidades não estejam compreendidas no escopo da linguagem.

Palavras chave: Corretor de código, Ammit, aplicação web, Ensino de programação

ABSTRACT

This work consists of a proposed improvement for how code correction software handle the generation of test cases. With the elaboration of its own interpreted language that uses Regular Expressions for content generation for the creation of test cases, the authors have developed a web application that provides users with an environment for registering questions and sending response attempts to serve as proof of concept for the language itself.

It is made possible during system use that users automatically generate test cases by use of the Ammit language or manually, should their needs not be comprised within the scope of the language.

Keywords: Code correction, Ammit, Web application, Programming teaching

Sumário

Lista de Figuras

1 Introdução

A educação no Brasil vivencia uma série de problemas que corroboram para o abandono escolar. Dentre os quais, a análise imprecisa (ou inexistente) do erro é apontada como um dos fatores causadores do fracasso escolar. Kutzke (2015) ressalta que não é o errar em si, mas a falta de atenção dada ao erro, que contribui para o insucesso dos alunos, e por consequência, do ensino.

No tocante ao ensino de programação, o próprio Kutzke (2015) propõe uma ferramenta para a análise do erro para exercícios de lógica de programação. Denominado FARMA-ALG (Ferramenta de Autoria para a Remediação de erros com Mobilidade na Aprendizagem - Algoritmos) em alusão à ferramenta FARMA (MARCZAL; DIRENE, 2012), a ferramenta permite aos professores criar exercícios de programação e disponibilizá-los aos alunos, corrige ditos exercícios com base em uma série de casos de teste, e permite ao professor e ao aluno uma visualização dos erros cometidos.

Embora funcione bem no que se propõe, o FARMA-ALG apresenta problemas que derivam da forma como os casos de teste são criados. Para cada questão, deve-se criar um ou mais casos de teste manualmente, o que é demorado e pode não cobrir o escopo da questão de maneira adequada. Além disso, os casos de teste são exibidos ao aluno, que pode então codificar uma instrução *switch case* com os valores de entrada e saída, apenas para satisfazer os casos de teste elaborados e assim apresentar uma solução aparentemente sem erros.

Visando sanar estes problemas, os autores propuseram a reestruturação no método de correção das questões, através da criação do Ammit, uma linguagem interpretada projetada para alteração da maneira que casos de teste são criados e avaliados. Primeiro, a geração de casos de teste se torna possível através de uma abreviada notação que descreve as entradas que o programa do aluno deve receber, possibilitando a geração automatizada de entradas para casos de teste; desta forma cada caso de teste que consumiria várias linhas pode ser descrito em uma linha, poupando tempo ao docente. E segundo, a avaliação da corretude de

cada resposta é feita submetendo as entradas geradas para cada caso de teste tanto ao algoritmo do aluno quanto a um algoritmo de solução-base fornecido pelo professor.

O sistema aqui descrito, homônimo à linguagem, é um conjunto reduzido da funcionalidade total do FARMA-ALG, provendo funcionalidades de cadastro de questões por usuários professores, e submissão de respostas por usuários alunos. A diminuição no escopo de funcionalidades visa prover uma prova de conceito voltada para a linguagem proposta, embora tenha-se incluído a opção de casos de teste manuais com a mesma sintaxe utilizada no FARMA-ALG, para complementar os casos em que Ammit não supre a necessidade. As alterações na forma como a correção é feita fornecerá resultados mais detalhados e a automação da criação de casos de teste reduzirá drasticamente o tempo que o docente precisará dedicar a um único exercício.

1.1 Problema

Foi argumentado por Kutzke (2015) que a falta de atenção ao erro pode lesionar todo o processo de ensino do aluno. Desta forma, toda ferramenta desenvolvida para auxiliar no referido processo deve buscar esta atenção ao erro e análise minuciosa do mesmo, para que sejam fornecidos dados precisos sobre o desempenho dos usuários da ferramenta.

Com a forma atual do software é possível que o aluno contorne a ferramenta de correção assim obtendo uma resposta dada como correta mas sem atingir o objetivo do exercício em sua totalidade. Ao continuar a executar um software com falhas desta natureza todo o processo educacional sofre, uma vez que o objetivo de recolhimento de dados precisos não é cumprido..

Docentes usuários do sistema também sofrem com uma deficiência de funções, uma vez que atual análise superficial do problema não permite que encontrem os erros mais comuns entre os alunos e planejem suas aulas de forma a abordar o problema.

1.2 Objetivos

O tópico a seguir aborda as soluções propostas pelos autores aos problemas encontrados e documentados no capítulo anterior. Tais soluções são então organizadas na forma de objetivos a serem cumpridos durante a elaboração do projeto.

1.2.1 Objetivo geral

O objetivo central deste projeto é a criação de uma linguagem para geração automática e parametrizada de casos de testes, e através desta demonstrar como a ferramenta FARMA-ALG pode ter seu uso simplificado para o professor através da redução do tempo dedicado à composição de um exercício.

Tal objetivo pode ser alcançado através da criação de uma aplicação web similar ao FARMA-ALG, doravante chamada de Ammit, que gera os casos de teste e processa a correção das submissões dos alunos.

1.2.2. Objetivos específicos

- Redesenhar a ferramenta de correção hoje embutida no FARMA-ALG
- Melhorar a ferramenta de correção hoje embutida no FARMA-ALG
- Criar a possibilidade de ensino orientado a correção do erro com base nos dados coletados pelo sistema.
- Diminuir o trabalho e tempo empregados pelo docente na criação de casos de teste.
- Formular o sistema de correção de maneira independente do FARMA-ALG para que possa ser utilizado para compilação e correção de códigos de várias fontes.

1.3 Justificativa

Apesar da existência de outras ferramentas para a correção de código, o Ammit tem como proposta facilitar o desenvolvimento de exercícios de programação mais elaborados desonerando o professor do tempo que seria gasto criando um caso de teste complexo o suficiente para avaliar ditos exercícios.

O ganho de tempo no desenvolvimento de casos de teste, além de seu valor educacional, diferencia o Ammit de outros corretores de código por usar sua própria linguagem para a geração de entradas de caso de teste, o que garante grande versatilidade à aplicação mesmo com um conjunto reduzido de operações.

A realização com sucesso da reformulação proposta tem como resultado a longo prazo melhorar todo o processo educacional dos alunos que usem o sistema, advogando assim em favor da utilização de ferramentas externas durante o ensino, ferramentas estas que alguns profissionais da área de ensino ainda hesitam em usar devido a dúvidas em relação a sua eficácia.

2. Fundamentação teórica

Em busca da reestruturação pretendida, os autores precisaram primeiramente desenvolver uma alternativa para a geração de casos de teste, alternativa esta que se deu através da criação de uma linguagem interpretada, em que se aplica conceitos de teoria da computação para a geração de conteúdo.

Uma vez a linguagem criada, se fez necessário o desenvolvimento de uma plataforma onde esta poderia ser empregada e testada, e posteriormente demonstrada. Em conjunto com as outras soluções propostas pelos autores, como a conversão da linguagem em uma API acessível por um *web service*, para que pudesse ser integrada com o Farma-ALG ou outras aplicações.

2.1 Autômatos Finitos

Um autômato finito, de acordo com Ricarte (2003), possui um conjunto de estados nos quais alguns são denominados como estados finais. A medida que caracteres de entrada são lidos o sistema passa de um estado a outro seguindo um conjunto de regras de transição que foram especificadas para o autômato. Após isto a validação para se a string de caracteres faz ou não parte da linguagem é dada através do último estado alcançado, se após o último caractere o estado for um dos estados finais a string então é reconhecida como parte da linguagem, caso contrário, a string não é reconhecida pelo autômato como parte da linguagem definida.

Autômatos podem ser representados de duas maneiras, note que o mesmo autômato é representado nas duas imagens: Através de um **grafo dirigido:**

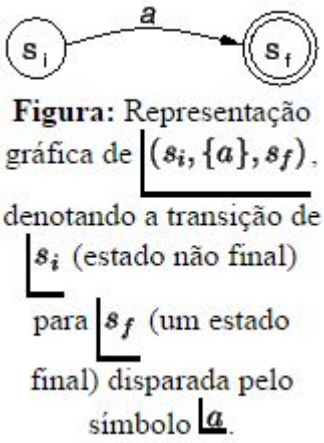


Figura 1: Representação em grafo dirigido
FONTE: Ricarte, Autômatos Finitos.

ou de **tabelas de transcrição:**

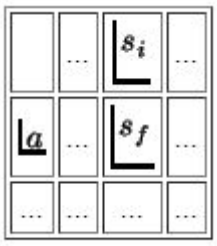


Figura 2: Representação em tabela de transcrição.
FONTE: Ricarte, Autômatos Finitos.

Na representação por grafos dirigidos os estados são representados por círculos (nós) e os estados finais são representados por círculos duplos e as transições por arestas rotuladas com os símbolos que disparam a transição entre os dois estados conectados.

A representação por tabelas de transcrição é considerada melhor para fins de processamento automático. Uma tabela de transição é uma matriz na qual as colunas representam os estados do autômato e as linhas os símbolos do alfabeto. Cada entrada na matriz indica qual o estado final de uma transição a partir do estado indicado na coluna através do símbolo indicado na linha.

Autômatos finitos são determinísticos quando para combinação de estado e entrada existe uma única transcrição aplicável, quando acontece o oposto o Autômato é chamado de Autômato finito não-determinístico.

2.2 Expressões regulares

O matemático norte-americano Stephen Cole Kleene desenvolveu as expressões regulares como uma forma de notação algébrica que ele chamava de **Álgebra de conjuntos regulares**. Esta notação então foi usada como base para os primeiros algoritmos computacionais de busca.

As expressões regulares são, por definição, uma forma concisa de definir uma cadeia de caracteres de interesse, essas sequências de caracteres são então interpretadas por um processador de expressões regulares que as traduz em uma representação interna que podem ser comparadas a uma string dentro de um código, usada então para pesquisa ou para substituição.

Exemplo de expressão regular:

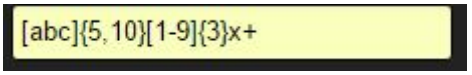


Figura 3: Demonstração de expressão regular

2.2.1 Generex

Generex é uma biblioteca apresentada como o oposto das expressões regulares comuns, uma vez que se propõe não a fazer a comparação entre uma expressão regular e uma string mas sim gerar uma string que é garantida de cumprir os requisitos estabelecidos dentro da expressão regular definida previamente. Ainda em desenvolvimento, atualmente o Generex não suporta todos os formatos de expressão regular disponíveis para uso em linguagens como o Java.

2.3 Análise léxica e sintática

O léxico compreende o dicionário implementado junto a linguagem e aplicação, sendo assim uma parte fundamental para a construção do sistema, assim como é para a maioria das aplicações. O dicionário é a estrutura de dados que compreende os itens lexicais e seus significados, os itens que constituem uma entrada lexical podem ser isolados ou um grupo de expressões organizadas para obter um novo significado. De forma comum cada entrada do léxico tem um significado semântico, no entanto, ocasionalmente podem ser atreladas associações à representações contextuais.

O léxico é fundamental para uma aplicação por definir os limites de o que está dentro de seu escopo ou não, existindo duas instâncias do léxico dentro da maioria das aplicações. Primeiramente tem-se o dicionário referente a linguagem, que é existente na aplicação desde o momento que a linguagem para seu desenvolvimento é definida, e este é referente à seus limites, a suas expressões únicas e suas palavras reservadas. Na sequência, durante o desenvolvimento da aplicação, é desenvolvida uma segunda instância do Léxico, que armazena termos e expressões definidos durante o desenvolvimento, como nomes de variáveis globais para toda a aplicação, classes e demais funções determinadas pelo desenvolvedor durante a criação da aplicação em questão.

A análise léxica consiste em decompor uma sentença em cada um de seus itens e então buscar o significado associado no léxico para cada um dos itens separadamente e então reunir os significados quando for necessário. Sua aplicação na computação e no desenvolvimento de sistemas está compreendida na interpretação de cada linha de código e expressão escrita pelo desenvolvedor, associando cada termo único da linguagem ao seu significado e propósito.

Quando aplicada a proposta do Ammit a análise léxica tem duas tarefas principais, sendo a primeira interpretar o que foi estipulado pelo professor na criação dos limites de um caso de teste e então passar essa informação adiante para a análise sintática. Sua segunda função é a própria geração e limitação do Léxico, ferramenta imperativa para a o funcionamento correto da geração automatizada de

casos de teste. Sem a análise léxica de cada limite estipulado pelo docente o restante do sistema não pode gerar automaticamente os referidos casos de teste.

Enquanto a análise léxica trabalha em nível com cada sentença separadamente, a análise sintática trabalha em nível de frases, recebendo sequências de expressões e então determina se a junção delas forma ou não uma frase na língua estabelecida. Poderá também construir uma árvore de derivação, que explicita as relações entre as palavras que compõem a sentença. O analisador sintático faz uso do léxico, que reúne o conjunto de itens lexicais da língua, e de uma gramática, que define as regras de combinação dos itens na formação das frases

2.4 Impacto da medição do erro através de ferramentas da informática Educacional

De acordo com Kutzke (2015) existe uma culpabilização do erro por parte dos professores que utilizam técnicas da Pedagogia Tradicional, que trata o erro como displicência do aluno com seus estudos ou até como falta de capacidade psico-biológica. Uma consequência possível levantada por Kutzke(2015) com base em Aquino (1997) é a evasão escolar resultante da análise superficial do erro e como consequência a já mencionada culpabilização do aluno.

Mesmo quando o professor busca fazer uma análise mais profunda do erro ele está fadado a encontrar certas dificuldades, algumas delas derivadas do volume de alunos que cada professor possui. Entre estas dificuldades é possível citar o tempo, uma vez que analisar minuciosamente cada exercício de cada aluno demandaria uma quantia absurda do mesmo, também se faz presente a dificuldade de fazer comparação entre os resultados obtidos para então serem obtidos dados como erros mais comuns. E quando usado o papel e formulários escritos, surgem problemas logísticos como o espaço para armazenar e a quantidade de registros de uma única turma.

A implementação da informática na educação, através de ferramentas como o FARMA-ALG e o Ammit, busca então tornar possível uma análise mais profunda

enquanto os problemas citados anteriormente são evitados. Ao implementar ferramentas como as citadas anteriormente o professor pode então analisar e agir sobre o erro a partir de uma abordagem diferente da tradicional culpabilização e assim tratar o problema antes que ele possa aumentar para finalmente resultar em consequências como a evasão escolar.

Junto a possibilidade de impedir o escalamento do erro em problemas maiores, a medição do erro através de ferramentas da informática educacional tem apresenta também outros pontos positivos como a melhora do processo de aprendizado como um todo, fornecendo ao professor informações como erros mais comuns entre os alunos para que finalmente o docente possa elaborar aulas e atividades que abordam com mais enfoque às dificuldades gerais de uma turma.

2.5 Sistemas de validação de código baseados em casos de teste.

Um caso de teste descreve uma condição particular a ser testada e é composto por valores de entrada, restrições para a sua execução e um resultado ou comportamento esperado. A utilização de casos de teste faz parte de sistemas desenvolvidos para correção de códigos em massa e softwares para a administração de competições, como o CD-MOJ (https://moj.naquadah.com.br/) e o BOCA (https://www.ime.usp.br/~cassio/boca/) , que são principalmente utilizados nos ambientes acadêmicos e de maratonas de programação.

A correção através de casos de teste é comumente executada em uma sequência simples: Uma vez definido o caso de teste, com suas saídas e entradas esperadas, o software que está efetuando o teste executa o código a ser testado com a entrada definida e então compara a saída obtida com a saída fornecida, fornecendo então um status de erro ou sucesso. Desta forma é possível garantir que o software que está sendo testado tenha seus comportamentos dentro do esperado,

Durante a utilização de grande parte dos sistemas existentes o caso de teste aceita apenas uma entrada e uma saída por caso de teste, gerando assim a

necessidade de gerar manualmente dezenas de casos de teste para cada condição na qual o sistema deverá ser testado.

3. Metodologia do Trabalho

O processo de desenvolvimento da linguagem Ammit, e consequentemente a aplicação de prova de conceito, foi realizado de acordo com as especificações do criador do FARMA-ALG, Alexander Robert Kutzke, que também atuará como cliente durante este projeto.

3.1 Modelo de processo de Engenharia de Software

Para o desenvolvimento optou-se pelo modelo ágil de Engenharia de Software chamado Kanban, que constitui da organização das atividades necessárias em quadros divididos em Para fazer, Fazendo e Feito.

O Kanban é caracterizado por permitir a visualização do fluxo de trabalho, limitar a quantidade de trabalho em andamento, Gerenciar e medir o fluxo de trabalho, tornar as políticas de processo explicitar e a utilização de modelos para reconhecer oportunidades de melhoria.

A utilização do Kanban traz como vantagem ao projeto a possibilidade de acompanhamento claro de quem está fazendo o que, quando começou e o quê já foi terminado, a facilidade de comunicação simplificada entre os autores, percepção visual do progresso do projeto e a simplificação do processo como um todo.

Durante a sua aplicação neste projeto a utilização do Kanban foi realizada através do aplicativo KanbanFlow (https://kanbanflow.com) que permite ao usuário criar vários quadros (boards) e fazer a gerência de atividades online através do mesmo. Para cada atividade descrita no plano de atividades que se encontra a seguir foi criado um novo quadro e então a atividade foi dividida em etapas menores e atribuídas entre os responsáveis.

3.2 Plano de Atividades

O plano de atividades a seguir foi dividido em três objetivos durante a sua elaboração e desenvolvimento do projeto, sendo estes a análise da tecnologia existente, a proposta e elaboração da linguagem como alternativa e por fim a elaboração da plataforma para apresentação e aplicação da mesma.

Objetivos.	**Atividades**	**Status**	**Responsáveis**
1 Etapa de análise para elaboração de alternativa para a geração de casos de teste.	1.1 Análise do sistema existente e entrevistar o desenvolvedor**.**	Concluído	Sergio Everton
	1.2 Análise de tecnologias existentes.	Concluído	Renam
	1.3 Elaboração da proposta alternativa para a geração automatizada de casos de teste e demais melhorias do sistema.	Concluído	Todos
2 Desenvolvimento da linguagem proposta para a automatização de casos de teste, a Ammit	2.1 Definição da sintaxe da linguagem Ammit	Concluído	Renam
	2.2. Definir regras comportamentais da linguagem Ammit e ajustes direcionados a atuação como seed para um autômato finito	Concluído	Todos
	2.3 Desenvolver a geração automatizada de casos de teste a partir da linguagem definida.	Concluído	Todos
	2.3 A partir da análise de requisitos e das definições comportamentais da linguagem propor a melhor forma de implementar a	Concluído	Everton

	linguagem em questão.		
3 Elaboração de uma aplicação web que permite o cadastro de questões, a criação de casos de teste, o envio de respostas, a aplicação de casos de teste à respostas enviadas e outras necessidades levantadas durante a análise de requisitos.	3.1 Criação da interface visual da aplicação.	Concluído	Sergio
	3. 2 Criação e modelagem do banco de dados	Concluído	Sergio
	3.3 Criação do cadastro e acesso separados para professores e alunos	Concluído	Todos
	3.4. Tornar possível o envio de arquivos durante a criação de casos de teste e durante o envio de respostas	Concluído	Todos
	3.5. Garantir que todo código enviado seja compilado antes de ser executado ou comparado.	Concluído	Todos
	3.6 Testes	Concluído	Todos

Tabela 01: Plano de atividades

3.3 Plano de riscos

Nº	Risco	Data Limite	Consequência	Ação	Dono do risco	Probab.	Impacto	Classif.
1	Falta de conhecimento.	19/04/2017	Atraso nas atividades, necessidade de treinamento	Definir tecnologias com as quais os autores já tenham familiaridade e quando não for possível, testes extensivos para aprendizado	Autores	Média	Médio	3

2	Cronograma não realista	15/06/2017	Impossibilidade de se manter fiel ao cronograma	Planejamento com folga para imprevistos, inicialmente sem contar fins de semana.	Autores e Orientador	Alta	Médio	4
3	Falta/saída de um membro da equipe	15/06/2017	Atraso nas atividades, redistribuição de responsabilidades	Alta comunicação para que todos os membros estejam igualmente cientes de todas as partes do projeto	Autores	Baixa	Alto	3
4	Falhas de comunicação	20/05/2017	Atraso nas atividades, problemas de integração do software completo	Fazer reuniões semanalmente e documentar tudo que for feito.	Autores	Média	Alto	4
5	Excesso de mudança de requisitos	20/05/2017	Atraso no cronograma e necessidade de retrabalho	Especificar requisitos e reafirmá-los em reuniões	Autores	Média	Alto	3

Tabela 02: Plano de riscos

Classificação	Valores
Muito Baixo	**1**
Baixo	**2**
Médio	**3**
Alto	**4**
Muito Alto	**5**

Tabela 03: Classificação de riscos

3.4. Desenvolvimento do projeto

Durante este tópico serão apresentadas as etapas pelo qual os autores passaram durante o desenvolvimento do projeto abordado, durante as etapas de análise e desenvolvimento.

3.4.1 Análise de Tecnologias

Este tópico abordará uma série de ferramentas que estão envolvidas de forma direta no desenvolvimento do Ammit ou o objetivo pretendido com a criação do *Web Service*

3.4.1.1 Generex

Generex é uma biblioteca para expressões regulares que age como o oposto das mesmas, uma vez que quando instanciado um novo objeto Generex vai receber uma string que esteja dentro dos parâmetros definidos pela expressão regular fornecida durante sua chamada. A expressão regular fornecida precisa ser enviada primeiro ao Automaton, cujo funcionamento será descrito na sequência.

Durante o instanciamento do novo objeto Generex podem ser definidos: A expressão regular como parâmetro para a string que irá ser gerada, os tamanhos mínimo e máximo da string que será gerada ou quantas strings serão geradas.

A biblioteca Generex foi utilizada para tornar possível a geração de strings para casos de teste, sendo a única porção que utiliza os conceitos de teoria dos autômatos.

3.4.1.1.2 Automaton

Automaton implementa autômatos finitos determinísticos e autômatos finitos não-determinísticos com suporte ao alfabeto Unicode e suporte a operações de expressões regulares, suporte este que inclui todas as operações padrões (Concatenar, unir, etc) e várias que são consideradas não-padrões (Complemento, intersecção).

A expressão regular usada como parâmetro para criação de novas strings é primeiro transformada em um autômato finito determinístico pelo Automaton, que então permite que ela seja usada como *Seed* para a geração das novas strings especificadas na instância do objeto.

3.4.1.2 ANTLR

ANTLR (ANother Tool for Language Recognition) é definido como um poderoso gerador de analisadores sintáticos utilizado para a criação de linguagens de programação, frameworks e outras ferramentas. Em desenvolvimento desde 1989 pelo professor de ciência da computação da Universidade de San Francisco, Terrence Parr, ANTLR é utilizado em diversas aplicações, como o ambiente de desenvolvimento integrado NetBeans, o framework de computação distribuída Hadoop, o software de análise legal Lex Machina, entre outros (Parr, 2012).

A partir de uma gramática construída com uma notação própria, embora similar à notação convencionada pelo Formalismo de Backus-Naur, ANTLR gera classes em Java ou C# descrevendo os analisadores léxico (*lexer*) e sintático (*parser*), bem como uma estrutura de *listeners* ou visitadores vazia, que podem ser estendidos pelo programador a fim de definir ações quando um determinado nó da árvore sintática (gerada em tempo de execução) é adentrado ou deixado (no caso dos *listeners*), ou visitado (no caso dos visitadores).

ANTLR comporta-se de maneira a ignorar eventuais erros de *parsing* decorrentes de uma instrução que esteja sintaticamente incorreta. Nas palavras do

autor (em tradução livre), ele se comporta como um ratel: “ele simplesmente não está nem aí”. Uma vez que a execução continua independente de *tokens* incorretos e/ou faltantes, é dever do programador antever os casos em que erros podem comprometer a devida interpretação/compilação da instrução dada.

Utilizamos ANTLR no desenvolvimento da linguagem Ammit, utilizada pelo sistema para geração de casos de teste aleatórios. Uma descrição aprofundada da linguagem é provida no capítulo 4.1.

3.4.1.3 TCC (Tiny C Compiler)

TCC é um compilador para a linguagem de programação C desenvolvido por Fabrice Bellard com a premissa de ser pequeno e extremamente rápido. Ao contrário de outros compiladores C, TCC é autossuficiente, dispensando a utilização de montadores e ligadores externos.

Utilizou-se o TCC para compilar os códigos-fonte para os exercícios sendo corrigidos, bem como os algoritmos corretores enviados pelos professores. TCC foi escolhido em detrimento da outra opção, GCC (*GNU Compiler Collection*) não apenas pela velocidade de compilação, mas também pelo espaço em disco ocupado (275Kb, versus 46Mb do GCC - embora isso se deva ao fato de GCC suportar outras linguagens como C++, Objective-C, Fortran, Go e Ada). Apresentamos este sistema como uma prova do conceito por trás da geração de casos de teste, portanto entendemos desnecessário adicionar suporte a múltiplas linguagens, embora seja fácil estender suporte no futuro.

3.4.2 Requisitos

O levantamento de requisitos para o desenvolvimento foi realizado através da técnica Análise do Sistema Existente. Durante a análise, o criador do sistema foi entrevistado em diversas ocasiões para que então fosse fornecida uma visão do

funcionamento do sistema e de suas limitações. Através desta técnica é possível identificar informações com relação ao fluxo de trabalho e atividades que compõem o sistema.

Esta técnica foi aplicada ao sistema em uso FARMA-ALG (http://ufpr.farma-alg.com.br/users/sign_in) para que fossem identificados os principais pontos de melhoria. Através da junção da análise e da entrevista com o desenvolvedor responsável pelo sistema, foram levantados dois pontos principais nos quais novas melhorias deveriam ser focadas primeiro. O primeiro desses pontos foi a forma como são criados casos de teste pelos professores, sendo assim uma melhoria voltada para que a experiência de docentes com o sistema melhore. O segundo ponto levantado foi a forma como as correções eram feitas, para que pudessem fornecer resultados mais precisos e detalhados, sendo assim uma melhoria voltada para a melhorar a experiência dos diferentes tipos de usuários do sistema em questão.

3.4.2.1 - Requisitos funcionais

Martin (2010) descreve requisitos funcionais como aqueles que definem o comportamento do sistema e são documentados através de casos de uso.

Durante o processo de análise do sistema existente foi possível captar as principais necessidades sentidas atualmente pelos usuários do sistema em ambas as funções (docentes e alunos). A partir dessa análise foi elaborada a lista de requisitos funcionais (Apêndice A) e com ela elaborado um planejamento de desenvolvimento.

3.4.2.2 - Requisitos não funcionais

Requisitos não funcionais também são chamados de suplementares, e constituem um conjunto que se aplica a toda a base do software em desenvolvimento e tem como uma característica que deve ser notada a sua

imprevisibilidade quando comparados com os requisitos não funcionais. Para o ammit foram levantados os seguintes requisitos como bases necessárias:

- Interface deve ser responsiva
- O uso contínuo do sistema deve trazer benefícios para professores e alunos.
- O sistema deve ser fácil de aprender.
- O sistema deve contar com ferramentas de relatórios e administração para melhoria do controle de erros como proposto durante sua problematização inicial.

3.4.3 Modelagem de dados

Tendo como base as informações adquiridas durante o levantamento de requisitos, sejam estes funcionais ou não funcionais, foi elaborada uma estrutura para o desenvolvimento do sistema, durante a qual foi definida a linguagem na qual ele seria desenvolvido, as alternativas para abordagem dos problemas encontrados durante a análise do sistema existente e também o melhor banco de dados para a realização do proposto.

Após a elaboração da estrutura citada se fez necessária a elaboração de diagramas que facilitam a compreensão sobre o fluxo de informações com o qual o sistema irá operar. Para isto foi necessário a elaboração dos seguintes diagramas: Diagrama de classes, Diagrama de sequências, Modelo Entidade Relacionamento e Diagrama Entidade relacionamento. Os diagramas foram anexados em seus respectivos apêndices.

3.4.4 Linguagem e ferramentas

Para o desenvolvimento do software foram usadas as seguintes tecnologias e ferramentas, cujas funções incluem não só a construção direta do software mas também a integração da equipe com ele através de controle de versões, e também

a interação da equipe entre si, para que as reuniões necessárias para manter o projeto alinhado acontecessem de forma dinâmica.

3.4.4.1 Java

Java é uma linguagem de programação orientada a objetos e de ampla utilização, estando presente em formas diferentes em bilhões de dispositivos, de carros a geladeiras.

A linguagem é executada através de uma máquina virtual chamada de JVM (Java Virtual Machine) que traduz os bytecodes em código de máquina. A execução através da JVM é o que permite que os códigos escritos em Java possam ser executados em qualquer sistema operacional sem a necessidade de alteração, sendo essa compatibilidade um dos grandes atrativos da linguagem.

Entre as especificações disponíveis do Java se encontra a JAVA EE (Java Enterprise Edition) que torna possível a criação de páginas web responsivas chamadas de jsp (JavaServer Pages) e a criação de servlets que são classes capazes de entender e tratar requisições através do protocolo HTTP e desta forma facilitar e tornar mais dinâmica a comunicação com o banco de dados. Neste projeto, como será detalhado posteriormente durante sua apresentação no capítulo 4, a linguagem Java foi utilizada para criação das interfaces e para a automação de todo o processo de geração de strings e comunicação com o banco de dados.

3.4.4.2 Netbeans

Netbeans é a IDE oficial para o Java 8, sendo construída através de um projeto open source colaborativo onde todos os desenvolvedores podem trabalhar para melhorar a IDE como um todo.

Como resultado da condução do projeto de forma open source novas ferramentas são constantemente lançadas para melhorar a experiência junto a IDE, atendendo as mais diversas necessidades. No entanto mesmo sua distribuição ‘simples’ contém ferramentas para auxiliar o desenvolvedor a desenvolver códigos mais limpos e sem bugs, ferramentas para facilitação de refatoração do código e uma ferramenta para a geração automatizada de TDDs, sendo estas ferramentas, juntamente com a familiaridade de todos os membros do projeto com a IDE, que constituem o principal motivo para a escolha do Netbeans como a IDE para o desenvolvimento do projeto.

3.4.4.3 Discord

Ferramenta gratuita que permite conferências de áudio entre vários participantes, o compartilhamento de arquivos, e a portabilidade da comunicação, uma vez que o usuário pode começar uma conferência no computador, se ausentar dela e então continuar na mesma conferência pelo celular, sem necessidade de refazer a conferência e sem perda de dados sejam esses textos ou arquivos compartilhados entre os participantes.

Sua utilização neste projeto se fez necessária para que reuniões a distância fossem possíveis entre os autores, permitindo através disto que fossem realizadas reuniões sempre que necessário para a discussão de quaisquer pontos de dúvida ou revisão durante o projeto. Durante o manifesto ágil é defendida a necessidade de comunicação constante e de qualidade entre a equipe, objetivo que foi atingido através do uso desta ferramenta.

3.4.4.4 Gantter

O Gantter é uma ferramenta gratuita para o planejamento de atividades dentro de um projeto através da criação de gráficos de Gantt. A proposta da ferramenta é a criação de cronogramas e planejamentos de atividades através da web, permitindo então que seus usuários acessem os cronogramas em qualquer lugar.

A escolha desta ferramenta é justificada por sua portabilidade, permitindo que os autores do projeto acessassem os cronogramas no celular ou na faculdade e com isso buscassem uma melhor qualidade para o projeto respeitando as metas e a administração do tempo.

3.4.4.5 Astah Community

O Astah é um software para modelagem de dados compatível com UML. Através da sua distribuição denominada **Community**, a distribuição usada nesse projeto, ele permite que de forma gratuita seus usuários possam criar e modelar diagramas de classe, diagramas de caso de uso e demais diagramas necessários para um melhor entendimento do projeto e de seu objetivo.

Sua escolha se deve ao fato de que através da distribuição escolhida é possível ter acesso à um software de reconhecimento internacional como um dos melhores para modelagem de forma gratuita, proporcionando uma plataforma intuitiva com diversas funções que permitem que a modelagem seja o mais precisa e clara possível, o que permitiu que os autores buscassem elaborar a melhor modelagem possível para auxiliar durante todas as etapas seguintes do desenvolvimento.

3.4.4.6 MySQL

MySQL é um sistema de gerenciamento de banco de dados objeto-relacional de código aberto mantido pela Oracle, a mesma empresa que mantém a linguagem Java.

Como um software o MySQL garante estabilidade na administração de bancos de dados, assim como suporte às funções e tipos de variáveis mais comuns da linguagem TSQL. Sua escolha para o desenvolvimento deste projeto se deve a familiaridade prévia de todos os autores com o software em conjunto com a grande confiabilidade e estabilidade disponibilizadas.

3.4.5 Testes

Para a validação do funcionamento do sistema foram executadas três etapas descritas e documentadas no tópico a seguir: Testes unitários, Testes de integração e testes de validação. Os testes descritos aconteceram durante períodos diferentes do desenvolvimento do projeto e foram executados mais de uma vez, os planos de testes a seguir documentados relatam os testes em sua última execução antes da revisão final para a entrega do projeto.

3.4.5.1 Teste unitário

Testes unitários são testes de funcionalidades específicas de forma isolada, sendo realizados em paralelo à etapa de desenvolvimento.

Para a realização de testes unitários foi utilizado o framework JUnit, que é uma ferramenta para a criação automatizada de testes unitários. Através deste framework foram testadas as funções independentes do sistema e da linguagem Ammit.

Abaixo, seguem exemplos de testes gerados através do framework para a validação de código. A documentação completa de testes unitários se encontra no Apêndice B

```
//TESTES PARA NINT
//Nint sem repetição

@Test //gerar um único inteiro sem especificação de limite (i.e. entre -32768 e 32767)
public void singleExprSimpleNint()
{
  int i=Integer.parseInt(get("i"));
  assertTrue(i<=32767 && i>=-32768);
}
@Test //gerar um único inteiro com limite mínimo 5 (i.e. entre 5 e 32767)
public void singleExprLowerLimitNint()
{
  int i=Integer.parseInt(get("i[5:]"));
  assertTrue(i<=32767 && i>=5);
}
@Test //gerar um único inteiro com limite máximo 5 (i.e. entre -32768 e 5)
public void singleExprUpperLimitNint(){
  int i=Integer.parseInt(get("i[:5]"));
  assertTrue(i<=5 && i>=-32768);
}
```

Figura 4: Exemplo de teste unitário

3.4.5.2 Teste de Integração

Testes de integração validam os comportamentos de funcionalidades conjuntas, como o professor criar uma questão e ela estar imediatamente disponível para gerência em uma funcionalidade separada do sistema.

Para os testes de integração foi desenvolvido um plano de teste (Apêndice C) com a estratégia de Testes de Fumaça, que constitui a junção de vários módulos do projeto em construções específicas e estas construções foram submetidas ao plano de testes relacionado.

3.4.5.3 Teste de Validação

Testes de validação acontecem imediatamente após a conclusão dos testes de integração e visam validar se o software funciona de modo esperado e se atende todos os requisitos funcionais apurados durante a etapa de análise.

Testes de validação são usualmente executados por usuários em ambientes controlados, para este fim, foi criado um plano de teste baseado nos requisitos funcionais e então foi solicitado à um usuário externo que testasse o *web service,* primeiramente de acordo com o plano de teste fornecido e depois de forma livre.

O plano de testes, com os resultados apurados pelo usuário encontram-se no Apêndice D.

4. Apresentação da Solução

Neste capítulo será apresentado o fluxo de informações durante a execução do Ammit, assim como suas funcionalidades e comportamentos específicos para cada tipo de usuário que estará utilizando o sistema.

4.1 Apresentação da linguagem Ammit

Quando da concepção da ideia de simplificar a escrita de casos de teste para o Ammit, uma das possibilidades consideradas foi a de usar uma notação generalista dos valores a serem alimentados ao algoritmo sendo avaliado. A notação homônima usada no Ammit foi concebida com este propósito, e evoluiu ao longo do desenvolvimento deste trabalho para uma pequena linguagem interpretada, capaz de gerar casos de teste diferentes a partir de uma mesma entrada. O objetivo desta seção é prover um guia da utilização da notação Ammit, bem como explicar sua estrutura.

Ammit foi desenhada para ser invocada em comandos de uma única linha, de modo a, por sua vez, gerar uma única linha de variáveis que podem ser passadas como argumento ou injetadas na entrada-padrão do programa a ser executado (à discrição do desenvolvedor da plataforma de correção). Cada instrução Ammit é composta de uma ou mais expressões separadas por ponto-e-vírgula. Considera-se uma *expressão* como a descrição de um tipo de variável e parâmetros a serem utilizados na geração de uma ou mais variáveis daquele tipo. A saída de uma instrução Ammit é uma string à qual são concatenados os valores gerados para cada expressão válida encontrada, separados por um único espaço.

O desenvolvimento da linguagem começa com a definição de uma *gramática*. Na definição de Chomsky (1956), uma gramática é a teoria da estrutura da linguagem em análise. Como tal, ela define as regras para a justaposição dos elementos da linguagem de forma não-ambígua, permitindo que diferentes leitores dela extraíam o mesmo significado. A ferramenta ANTLR (referenciada no capítulo

3) descreve uma metalinguagem para criação de gramáticas para posterior geração dos analisadores léxico e sintático da linguagem.

```
grammar Ammit;
row      : expr (';' expr)* ;
expr     : nint | nstr | nconst;
nint     : 'i' ('[' intrange (',' repeat)? ']')? ;
nstr     : 's' '[' strregex ']' ;
intrange : (min)? ':' (max)?;
min      : sigint;
max      : sigint;
nconst   : sigint;
repeat   : INT;
sigint   : ('+' | '-')? INT ;
strregex : STR ;
INT      : [0-9]+ ;
STR      : '"' ~ ["\r\n]* '"' ;

WS : [ \t\r\n]+ -> skip ;
```

Figura 5: A gramática do Ammit

A gramática do Ammit descrita pela meta-linguagem do ANTLR é exibida na figura A.1. Cada linha descreve um símbolo (em inglês, *token*) a ser analisado pelo analisador léxico (tokens descritos com inicial maiúscula) e sintático (os com inicial minúscula). ANTLR pode converter dita gramática para implementações dos analisadores tanto em linguagem Java quanto C++; por isso, uma das regras na escolha da nomenclatura é não usar palavras reservadas da linguagem alvo (como *int* no Java; a utilização de *INT* não causa problema).

A partir da figura A.1, podemos perceber os elementos que compõem uma instrução Ammit válida. Como mencionado, cada instrução inteira de Ammit é composta por uma única linha, representada pelo símbolo *row*. Cada linha, ou *row*, é composta de uma instância do símbolo *expr* (a expressão mencionada anteriormente), seguida de zero ou mais sequências de ponto-e-vírgula e *expr*. Por sua vez, cada *expr* é por sua vez definido como uma instância de um dos três tipos suportados atualmente pelo Ammit: *nint* (novo inteiro), *nstr* (nova string) ou *nconst* (nova constante); e assim por diante, até chegarmos aos *tokens* do analisador léxico, *INT* (inteiro) e *STR* (string).

Uma vez definida a gramática, o ANTLR cria classes que invocam métodos do próprio ANTLR para a devida interpretação de cada comando. Em tempo de execução, uma árvore de análise como a da figura A.2 é criada pelo ANTLR e por ele percorrida em pré-ordem (isto é, o nó raiz é visitado, e cada filho da esquerda pra direita é recursivamente visitado); para cada nó, uma função de *callback* específica para o símbolo naquele nó é invocada. Na geração das classes, é possível optar entre um visitador (*visitor*) e uma escuta (*listener*), gerando as classes BaseVisitor e BaseListener, respectiva e mutuamente exclusiva, que criam instâncias vazias destes callbacks. A diferença é que o visitador acionará o *callback* no momento em que o nó é visitado, enquanto a escuta acionará métodos de *callback* em dois momentos: no caminho raiz-folha, e no caminho folha-raiz (*grosso modo*, quando o ANTLR “entra” e “sai” da subárvore que tem o nó em questão como raiz).

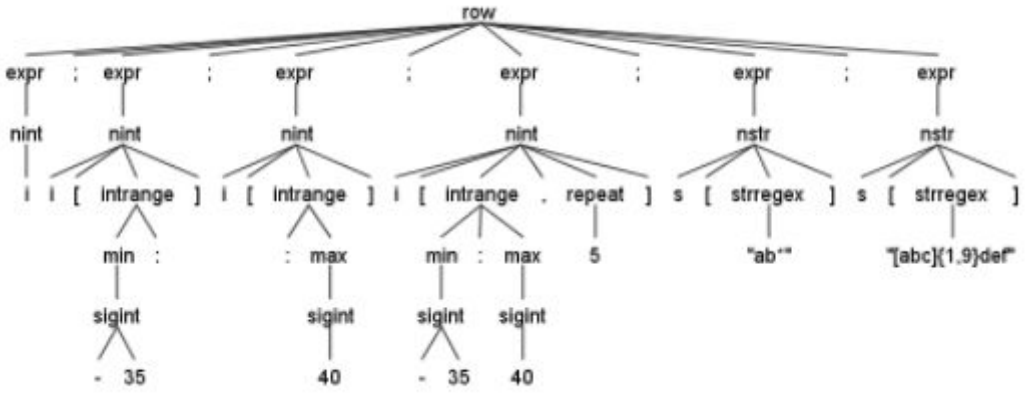


Figura 6: Árvore sintática.

No Ammit, optamos por utilizar uma especialização de BaseListener, que, segundo Parr (2012), é ideal para tradutores e interpretadores. A escolha foi feita em especial por dar a oportunidade de iniciar variáveis de controle no momento que a classe percorredora (ParseTreeWalker) entra no nó mais externo, e processar cada símbolo à medida que subimos na árvore.

4.1.1 - Sintaxe

Por ocasião da publicação deste trabalho, Ammit suporta inteiros de dois bits (ou seja, valores entre -32768 e 32767), strings, e constantes inteiras. A sintaxe para cada elemento é dedutível a partir da gramática da figura A.1, uma vez que se conheça a metalinguagem ANTLR (embora ela não difere muito de uma expressão regular tradicional), mas por questão de conveniência é explicada abaixo.

A sintaxe para inteiros consiste da palavra reservada “i”, seguida ou não por mais parâmetros. Quando nenhum parâmetro é dado, Ammit gerará um único inteiro entre -32768 e 32767 (os limites mínimo e máximo para inteiros de dois bits). Escolhemos inteiros de dois bits por questões de retrocompatibilidade com compiladores antigos que podem não ter suporte a inteiros de quatro bits (muito embora o compilador usado no sistema Ammit os suporte). Os parâmetros suportados, dados entre colchetes, são uma designação de intervalo mínimo e máximo para gerar entradas, e um inteiro positivo opcional que indica o número de inteiros a se gerar, separado por vírgula do intervalo. Assim, a instrução

```
i;i[1:100, 5];i[-100, -1]
```

gerará um inteiro qualquer de dois bits, e em seguida, cinco inteiros entre um e cem (inclusive), seguida de um inteiro entre -100 e -1. Qualquer um dos índices do intervalo pode ser omitido (até mesmo ambos), situação em que são usados os limites mínimo e/ou máximo de inteiro de dois bits. Ammit espera que os índices mínimo e máximo apareçam nesta ordem, mas os inverterá caso um índice máximo seja menor que o índice mínimo, com efeito, *i[100:1]* produz um inteiro entre 1 e 100, da mesma forma que *i[1:100]*. A sintaxe para constantes inteiras é similar, exceto que basta inserir o número diretamente, ou seja, a instrução

```
42;i[1:10, 42]
```

gera na entrada o número 42, seguido de 42 inteiros de um a dez.

Para strings, a geração é feita a partir de uma expressão regular (do inglês *regular expression*, abreviada *regex*) utilizando a biblioteca Generex (referenciada no capítulo 3). Assim, a sintaxe básica é a palavra reservada “s”, seguida da regex entre colchetes e aspas duplas. Muito embora os colchetes ou aspas pudessem ser dispensados, escolheu-se esse formato visando padronização e a facilidade de extensão futura. Assim, a expressão

s[“ab[cd]{3,7}e?”]

gera uma string de 5 a 10 caracteres, começada por ab, com três a sete ocorrências dos caracteres c ou d, e encerrada ou não com um e. Quaisquer expressões válidas podem ser passadas, exceto expressões utilizando aspas duplas (isto é, Ammit não suporta caracteres de escape). Ademais, a biblioteca Generex não suporta elementos de expressão regular que poderiam resultar num número infinito de strings, como por exemplo o asterisco (*).

4.2 Funções comuns.

Algumas funções são disponibilizadas antes da autenticação do usuário e desta forma são executadas de forma independente à um controle de sessão que identifica o tipo de usuário.

4.2.1 Acesso à aplicação

A tela de login é a primeira apresentada ao usuário, a partir dela ele pode preencher os campos e entrar com um usuário já cadastrado ou selecionar para cadastrar um novo usuário.

A passagem por esta tela é obrigatória para o acesso às funções subsequentes do sistema.

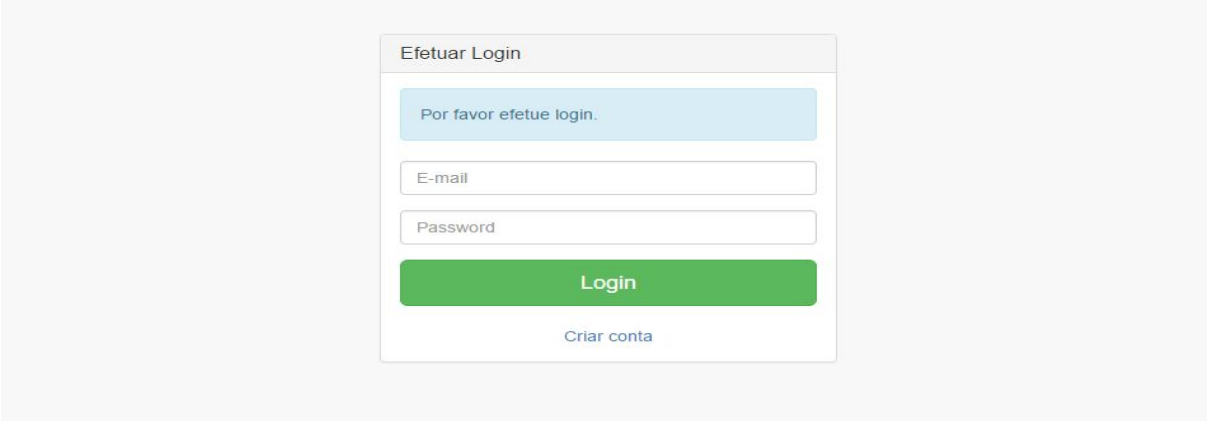


Figura 7: Tela de login

4.2.2 Cadastro de usuário

Se durante a tela de login o usuário optar por efetuar o cadastro de um novo usuário ele será então redirecionado a uma tela de cadastro onde preencherá as informações para a criação da conta e selecionará o tipo de conta a ser criada.

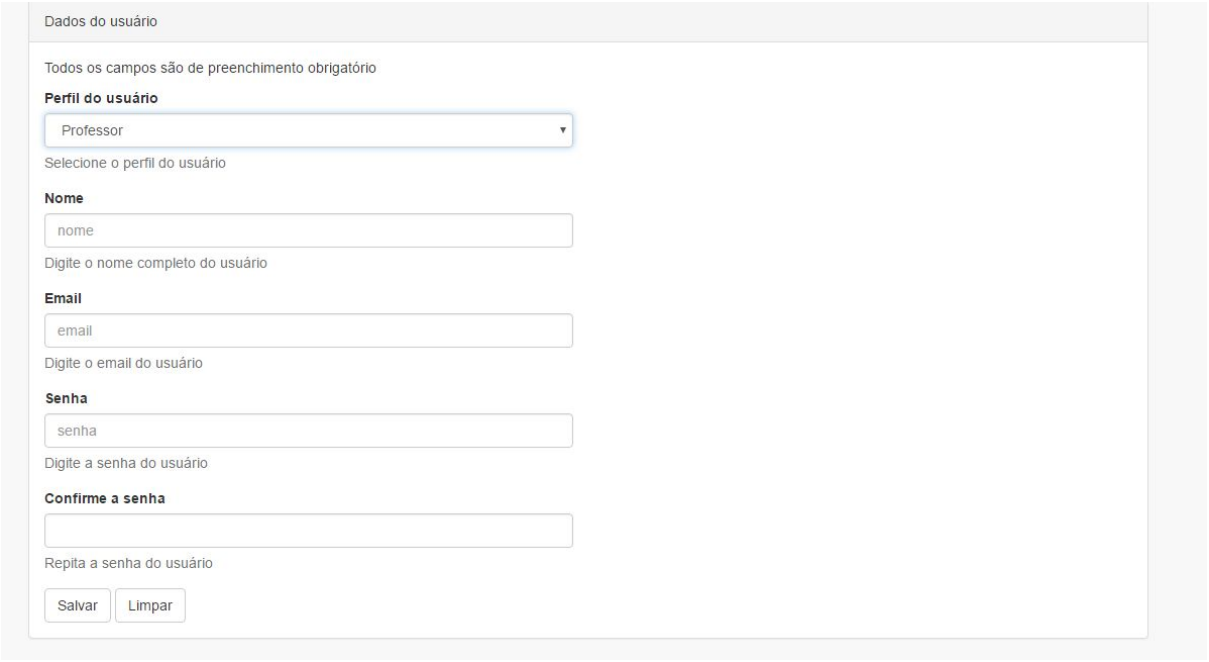


Figura 8: Criação de conta

4.3 Interface para o aluno

Quando a conta iniciada for identificada pelo sistema como pertencente à um aluno, algumas funções específicas serão disponibilizadas para ele. Desta forma, é criada uma interface responsiva que só fornece ao usuário o necessário, sem sobrecarregá-lo com informação inútil ou a qual ele não tem acesso.

4.3.1 Acessando questões

Com sua sessão iniciada o aluno poderá acessar todas as questões disponíveis para ele ou filtrá-las de acordo com um professor específico. A partir de uma lista com os títulos o aluno pode então acessar uma questão específica e ver mais detalhes sobre a mesma.

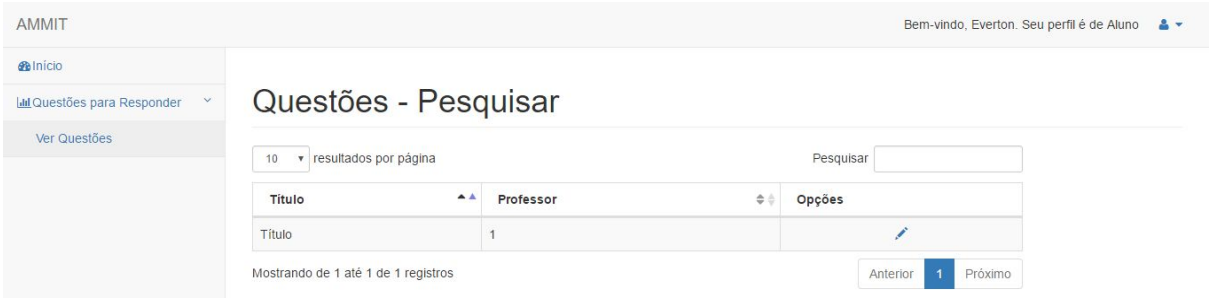


Figura 9: Lista de questões

4.3.1.2 Submetendo questões para correção

Uma vez que o aluno abra a janela com mais detalhes da questão escolhida, através da opção “Responder” no menu opções, ele poderá então submeter uma resposta para avaliação automática (Figura n). Em alguns segundos o sistema então retorna se a resposta está correta ou não de acordo com parâmetros de correção definidos pelo professor que criou a questão. (Figura X+2).

Figura 10: Cadastrar resposta

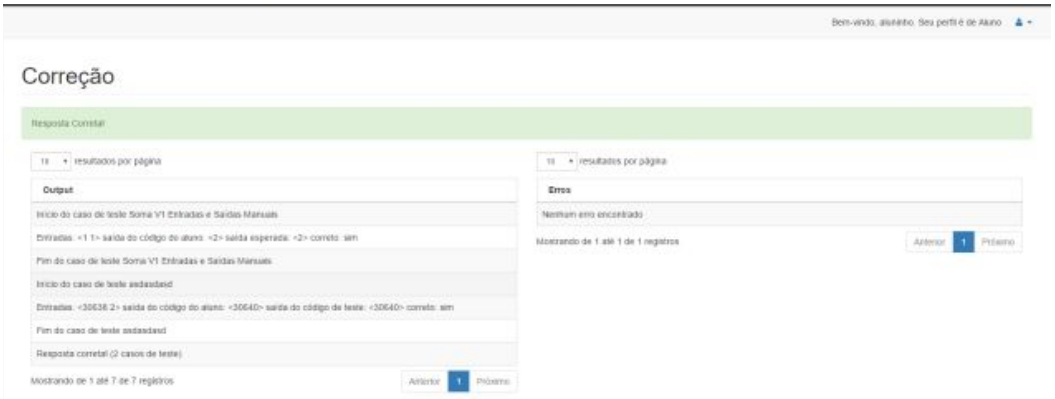


Figura 11: Resultados

4.4 Interface para o professor

Quando o sistema reconhece a sessão iniciada como pertencente a um professor, ele disponibiliza funções de criação e administração de questões, podendo alterar as questões e suas soluções de acordo com sua necessidade.

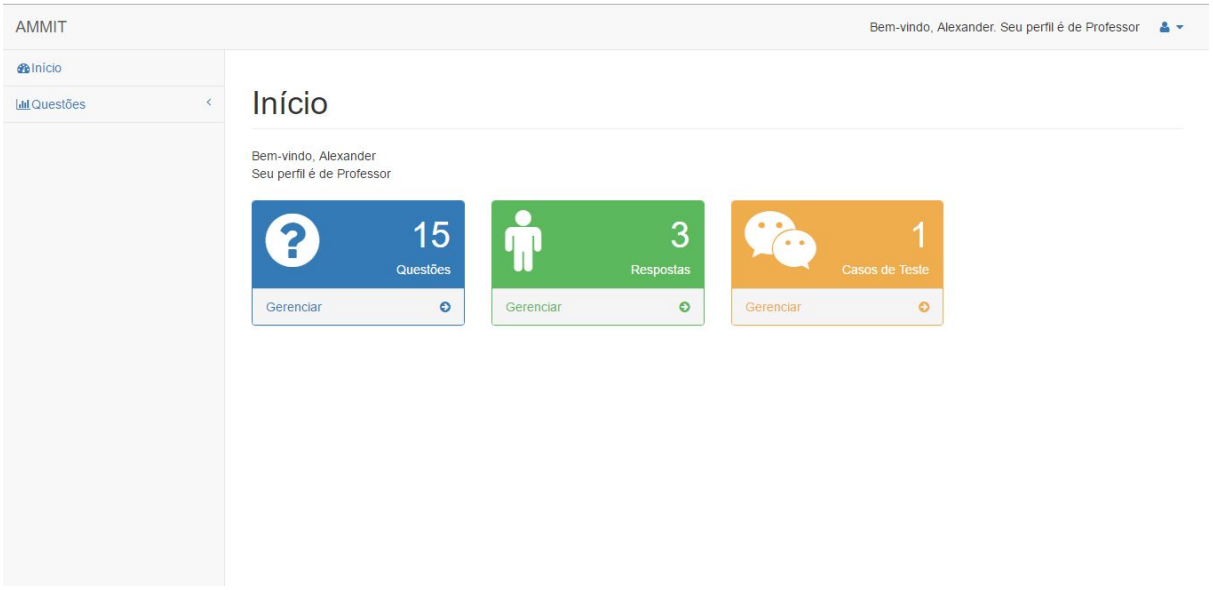


Figura 12: Tela inicial do professor

4.4.1 Criando questões

Função que permitirá ao professor criar questões que serão acessadas por alunos posteriormente.

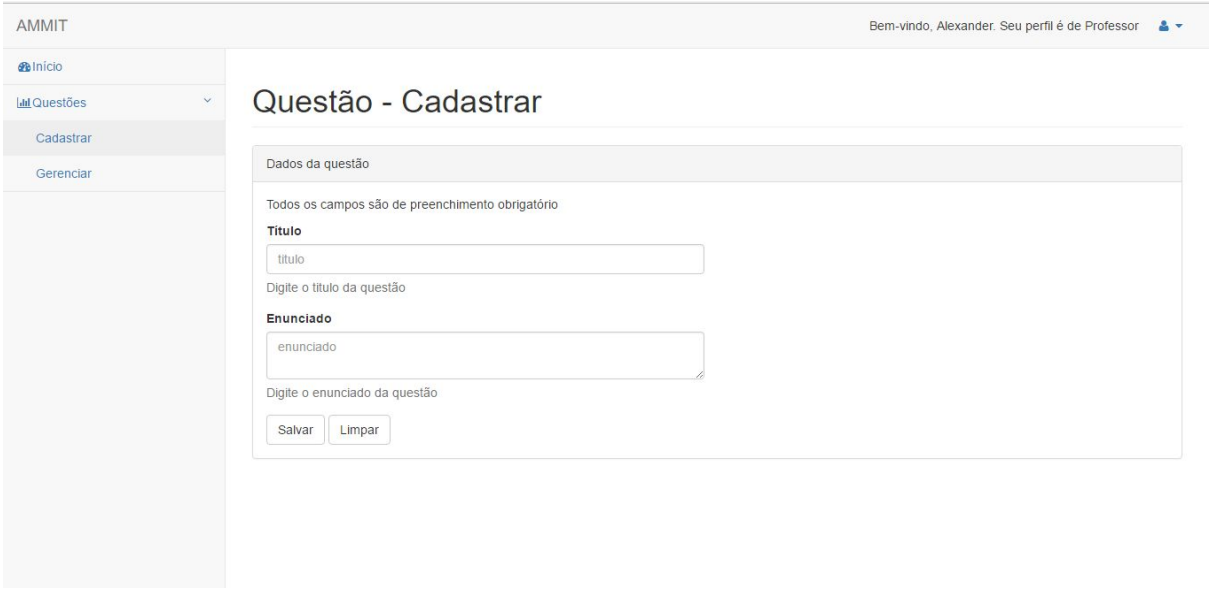


Figura 13: Criação de nova questão

4.4.1.2 Definindo casos de teste

Figura 14: Gerência dos casos de teste

4.4.1.2.1 Entradas

Durante a criação de questões o professor tem duas opções para a criação de entradas nos casos de teste para a correção da mesma: Utilizando a linguagem construída através do Ammit para que ele gere os conjuntos de entrada e saída automaticamente, e neste caso, para proporcionar uma melhor utilização do professor também é fornecido um breve tutorial de utilização da linguagem citada.

Figura 15: Entrada com a linguagem Ammit

Figura 16: Menu de ajuda da linguagem Ammit

O docente também tem a alternativa de determinar os casos de teste manualmente, caso precise de um caso em especial ou de algo que não está dentro do escopo da linguagem para geração automatizada.

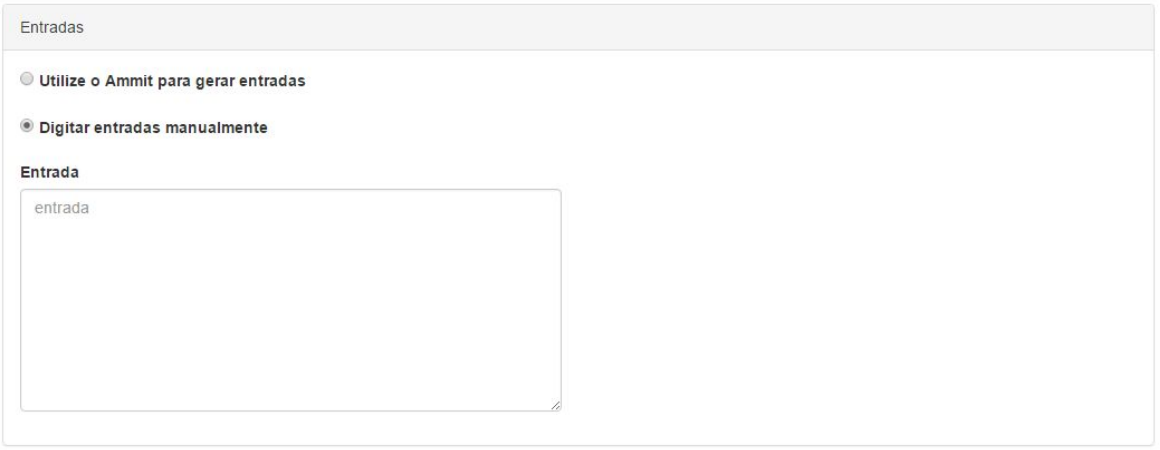


Figura 17: Entrada manual

4.4.1.2.2 Saídas

Para a validação de casos de teste, com suas saídas de dados, o docente também possui duas opções para a correção: Ele pode enviar um código desenvolvido para corrigir os códigos dos alunos ou pode definir manualmente as saídas esperadas quando as entradas determinadas de forma manual forem inseridas no código do aluno.

É importante atentar no entanto que caso a opção de utilizar o Ammit seja escolhida, o professor só pode enviar um código de validação, sem poder definir manualmente as saídas esperadas.

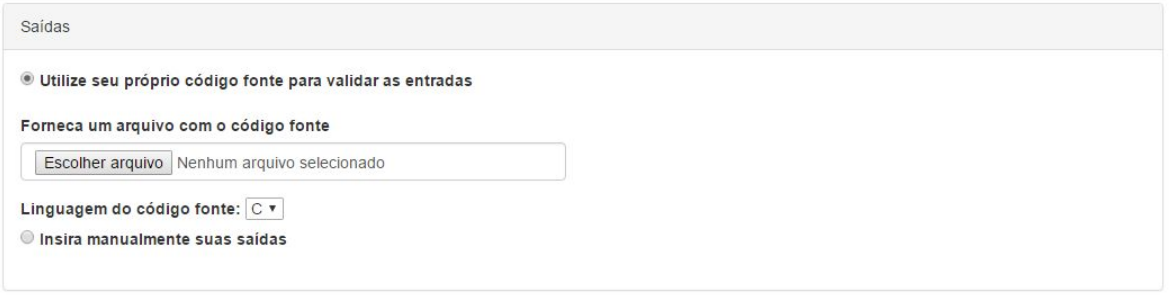


Figura 18: Definição de saída com envio de arquivo.

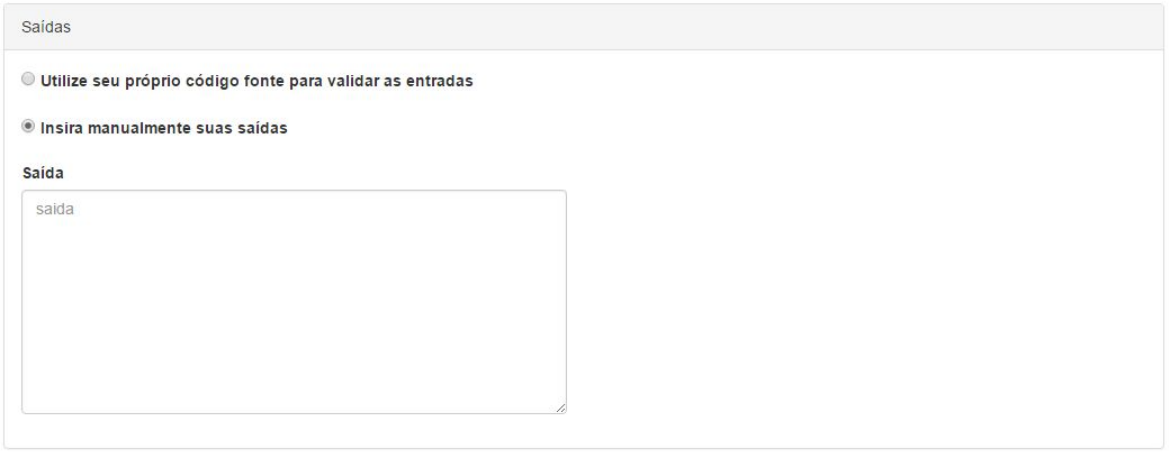


Figura 19: Definição de saída manualmente

4.4.2 Gerenciar questões

Ao acessar a interface destinada ao gerenciamento de questões o professor conta com uma lista de todas as questões criadas por ele assim como um campo de pesquisa para que possa encontrar mais facilmente a questão desejada.

Uma vez que a lista está carregada são fornecidas três opções para o professor: **Edição de título e enunciado**, **Edição de casos de teste** e **Deletar questão.**

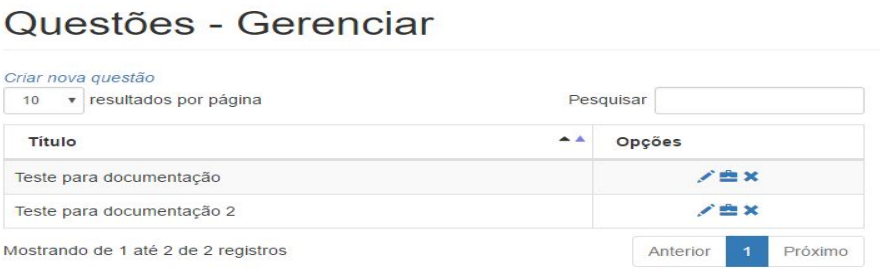


Figura 20: Gerência de questões

Figura 21: Edição de título e enunciado

Figura 22:Edição de casos de teste

Figura 23: Deletar questão

5. Considerações finais

Para o desenvolvimento deste trabalho, procurou-se aplicar os conceitos e técnicas aprendidos no curso de Tecnologia em Análise e Desenvolvimento de Sistemas, bem como práticas aprendidas no meio profissional, como a utilização de Kanban e a combinação de métodos para atingir um resultado que funcionou para a equipe. Conseguimos também incorporar, ainda que em pequena escala, conceitos teóricos que extrapolam a grade curricular do curso, como princípios de construção de compiladores, estrutura de linguagens de programação e teoria dos autômatos.

Obteu-se uma experiência que equiparamos a uma start-up; fomos nós os responsáveis pela definição de escopo e levantamento de requisitos para um produto final que acreditamos poder ser útil a várias pessoas que talvez não tenham pensado nessa possibilidade. Enfrentamos dificuldades de projeto, tivemos que pesquisar por ferramentas e técnicas adequadas à melhoria que desejávamos propor, e muito embora a linguagem Ammit ainda tenha um bom espaço para ser desenvolvida (como a cobertura de todos os tipos primitivos), o resultado foi bastante satisfatório.

Um ponto observado como negativo foi que, por fatores associados à comunicação, acabamos derivando da proposta inicial de implementar a linguagem já como uma API. Chegamos a um ponto do desenvolvimento em que retornar ao objetivo principal seria mais oneroso que readequar os objetivos do projeto para um fim tangível, assim relegando a proposta inicial a um trabalho posterior.

Um dos estudos que consideramos que pode ser feito para a melhoria contínua da avaliação de programação é determinar quantos casos de teste, idealmente, se deve rodar para se ter uma noção de quão correto ele está. Temos ciência de que (como lecionado ao longo do Curso) é impossível testar cem por cento de um software, mas deve ser possível chegar a um valor ótimo em tempo factível. Esse estudo foi posto de lado, em parte por não combinar com a proposta de trabalho que acabou por nós adotada, em partes pelo tempo de pesquisa que seria demandado para se obter dados concretos. Contudo, acreditamos que com o Ammit em seu estado mais maduro, possa ser uma ferramenta que permita alavancar estudos similares.

6. Referências Bibliográficas

Parr, T. The Definitive ANTLR 4 Reference.Dallas:The Pragmatic Programmers LLC, 2012. 305 p.
Parr,T.(eal).Getting Started With ANTLR v4. Disponível em <https://github.com/antlr/antlr4/blob/master/doc/getting-started.md>. Acesso em 04/06/2017.

Bellard, F (et al). Tiny C Compiler - Summary. Disponível em <http://savannah.nongnu.org/projects/tinycc>. Acesso em 15/04/2017

GNU.org. GCC, the GNU Compiler Collection. Disponível em <https://gcc.gnu.org/>. Acesso em 15/04/2017.

FANTINATO, M. AutoTest – Um framework reutilizável para a automação de teste funcional de software, 2005. Disponível em:

<http://comunidade.cpqd.com.br/cadernosdetecnologia/Vol1_N1_jan_dez_2005/pdf/artigo09_Fantinato.pdf> Acesso em 01/05/2017.

Ricarte, L. M. I. Autômatos Finitos, 2003 . Disponível em:
<http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node46.html> Acesso em 03/04/2017.

Generex. Disponível em:
<https://github.com/mifmif/Generex> Acesso em 02/03/2017.

ANTLR. Disponível em:
<http://www.antlr.org/> Acesso em: 18/04/2017.

Automaton. Disponível em:
<https://mvnrepository.com/artifact/dk.brics.automaton/automaton/1.11-8> Acesso em: 02/03/2017.

Tiny C Compiler. Disponível em:
<https://bellard.org/tcc/> Acesso em: 14/05/2017.

Netbeans. Disponível em:
<https://netbeans.org/> Acesso em: 10/03/2017.

Java. Disponível em:
<https://www.oracle.com/java/index.html> Acesso em 10/03/2017.

GUEDES, G. T. A. UML2: guia de consulta rápida. São Paulo: Novatec, 2005.

Ricarte, L. M. I. Introdução à Compilação. Rio de Janeiro: Elsevier, 2008.

Chomsky, N. Three Models for the Description of a Language. Cambridge, Estados Unidos:IEEE Transactions, 1956.

Kutzke, R. A. Informática educacional e a mediação do erro na educação: um estudo teórico-crítico e uma proposta de instrumento computacional, 2015

8. Apêndices

Apêndice A: Requisitos funcionais (casos de uso).

Durante o levantamento de requisitos funcionais as funções foram divididas entre os dois atores principais: Professor e aluno. Em conformidade com essa divisão encontram-se a seguir os diagramas de casos de uso detalhando o fluxo de dados para que cada função seja acessada pelo seu respectivo ator, juntamente com uma listagem de funcionalidades esperadas para cada ator.

A.1 Requisitos do professor.

Os requisitos atrelados ao professor estão relacionados com a administração de questões, administração esta que envolve a pesquisa, criação, edição e exclusão de questões.

O sistema deve permitir ao professor:

- Pesquisar questões
- Cadastrar questões:
    - Pesquisar casos de teste
    - Criar casos de teste
    - Fazer Upload de arquivo corretor
    - Alterar caso de teste
    - Remover casos de teste
- Alterar questões.
- Excluir questões
- Remover respostas enviadas.

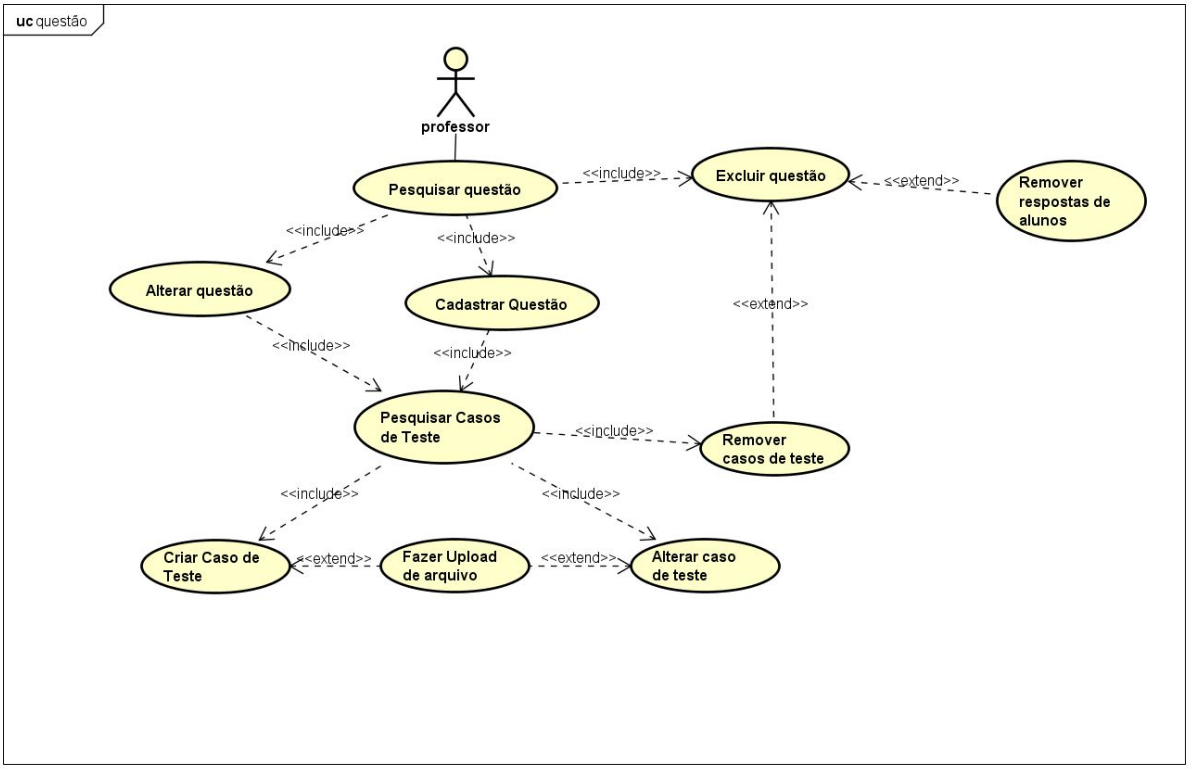


Figura A.1: Diagrama de caso de uso para professor

A.2 Requisitos do aluno:

Os requisitos atrelados ao aluno estão relacionados com o envio de respostas, assim como ao recebimento de correções efetuadas pelo sistema.

O sistema deve permitir ao aluno

- Pesquisar questões
    - Responder questões
    - Efetuar envio de arquivo
    - Receber avaliação da resposta enviada.

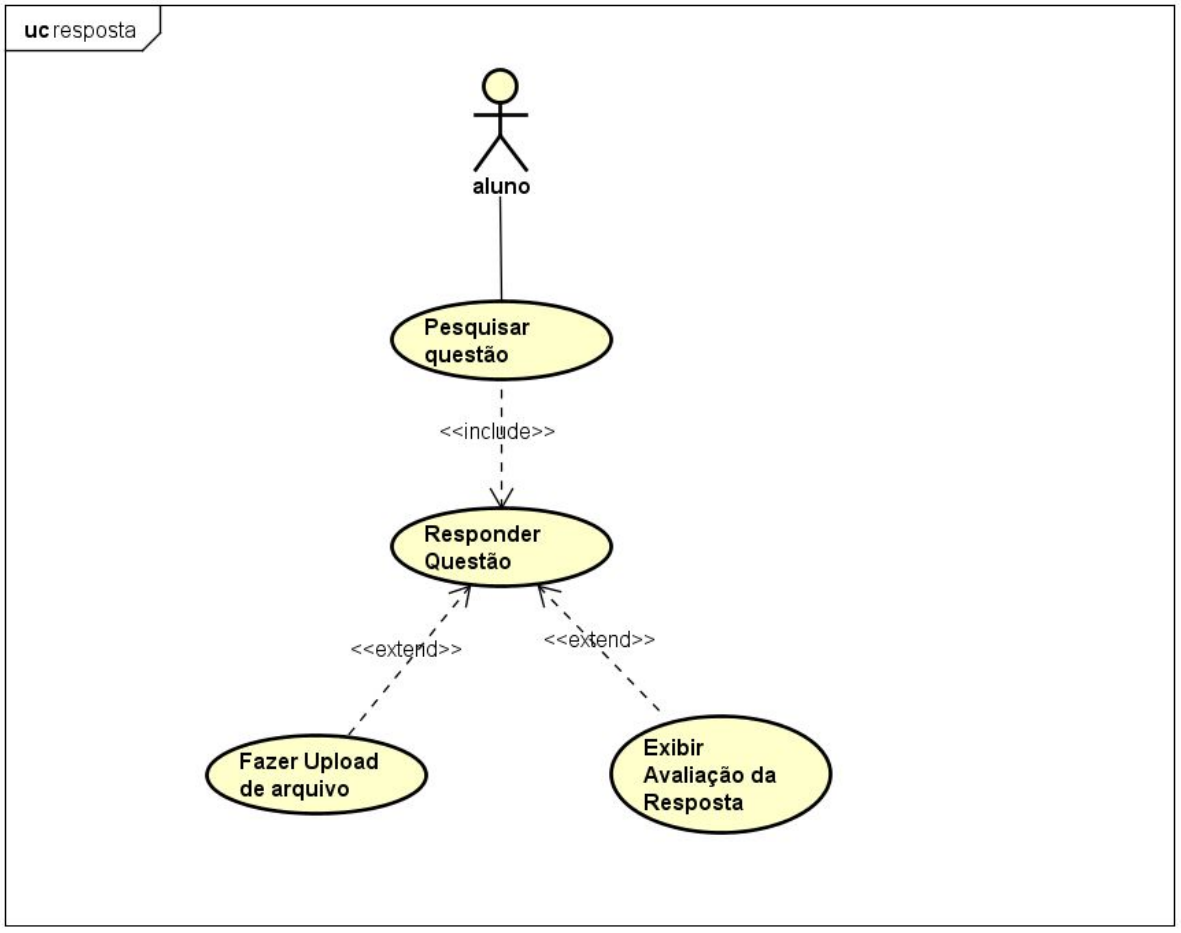


Figura A.2: Diagrama de caso de uso para aluno.

Apêndice B: Testes Unitários da linguagem.

Neste apêndice encontra-se o código fonte utilizado para o teste das funcionalidades da linguagem própria Ammit desenvolvida ao longo deste projeto. Como explicado anteriormente os testes em questão foram realizados utilizando JUnit.

```
“public class InputGeneratorTest {

    public InputGeneratorTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    //mascara a geração de uma string de teste
    private String get(String seed){
        return new InputGenerator(seed).generate();
    }

    /* Os testes abaixo testam o "caminho feliz" do AMMIT */

    //TESTES PARA NCONST
    @Test //gerar uma constante igual a 42
    public void singleNconst(){
        assertEquals(Integer.parseInt(get("42")),42);
    }
    @Test //gerar duas constantes, 35 e 29, separadas por um espaço
    public void twoNconst(){
        assertEquals(get("35;29"), "35 29");
    }
    @Test //gerar uma constante igual a -42
    public void singleNegativeNconst(){
        assertEquals(Integer.parseInt(get("-42")),-42);
    }

    //TESTES PARA NINT
    //Nint sem repetição
```

```
    @Test //gerar um único inteiro sem especificação de limite (i.e. entre -32768 e 32767)
    public void singleExprSimpleNint(){
        int i=Integer.parseInt(get("i"));
        assertTrue(i<=32767 && i>=-32768);
    }
    @Test //gerar um único inteiro com limite mínimo 5 (i.e. entre 5 e 32767)
    public void singleExprLowerLimitNint(){
        int i=Integer.parseInt(get("i[5:]"));
        assertTrue(i<=32767 && i>=5);
    }
    @Test //gerar um único inteiro com limite máximo 5 (i.e. entre -32768 e 5)
    public void singleExprUpperLimitNint(){
        int i=Integer.parseInt(get("i[:5]"));
        assertTrue(i<=5 && i>=-32768);
    }
    @Test //gerar um único inteiro com limites mínimo 0 e máximo 5 (i.e. entre 0 e 5)
    public void singleExprBothLimitsNint(){
        int i=Integer.parseInt(get("i[0:5]"));
        assertTrue(i<=5 && i>=0);
    }
    @Test //gerar um único inteiro com limite mínimo -5 e máximo -1 (i.e. entre -5 e -1)
    public void singleExprNegativeLimitNint(){
        int i=Integer.parseInt(get("i[-5:-1]"));
        assertTrue(i<=-1 && i>=-5);
    }
    //fim Nint sem repetição

    //Nint com repetição
    @Test //gerar entre 5 e 50 inteiros aleatórios todos entre 1 e 100
    public void minFiveMaxHundredRandomInts(){
        int total=0, qtd=new Random().nextInt(46)+5;
        String s=get("i[1:100,"+qtd+"]");
        String[] ints = s.split(" ");
        for(String next : ints){
            int i=Integer.valueOf(next);
            if(i<=100 && i>=1)
                total++;
        }
        assertEquals(total, qtd);
    }
    @Test //gerar zero inteiros
    public void zerorandomInts(){
        assertEquals(get("i[1:100,0]"),"");
    }



    /* Fim dos testes para o "Caminho Feliz" */


     /* Os testes abaixo tentam "quebrar" a linguagem e demonstram o comportamento da linguagem
ante a erros de sintaxe*/

    //TESTES PARA TOKENS EXPR INCORRETOS
    @Test //tenta gerar um inteiro usando sintaxe incorreta (omitido o "i" inicial): esperado string vazia
    public void singleNintMissingLeadingI(){
        assertEquals(get("[1:100]"), "");
    }
```

```
    @Test //tenta gerar cinco inteiros usando sintaxe incorreta (omitido o "i" inicial): esperado string vazia
   public void fiveNintMissingLeadingI(){
      assertEquals(get("[1:100,5]"), "");
   }
   @Test //tenta gerar um número negativo de inteiros: esperado o módulo do número
   public void negativeAmountOfInts(){
      int total=0, qtd=new Random().nextInt(46)+5;
      String[] s=get("i[1:100,-"+qtd+"]").split(" ");
      for(String next : s){
         int i=Integer.valueOf(next);
         if(i<=100 && i>=1)
            total++;
      }
      assertEquals(total, qtd);
   }
    @Test //tenta gerar múltiplos inteiros sem especificar a quantidade: esperado nenhum número gerado
   public void noQtdMultipleIntegers(){
      assertEquals(get("i[1:100,]"), "");
   }

   @Test //tenta gerar múltiplos inteiros sem vírgula: esperado um número gerado
   public void noCommaMultipleIntegers(){
      assertEquals(get("i[1:100 5]").split(" ").length, 1);
   }

    @Test //tenta gerar inteiro onde o limite mínimo é maior que o máximo: esperado inteiro gerado normalmente
   public void minBiggerThanMax(){
      int i=Integer.valueOf(get("i[100:1]"));
      assertTrue(i>=1 && i<=100);
   }

   @Test //tenta gerar inteiro sem especificar os limites
   public void missingBothMinAndMax(){
      int i=Integer.valueOf(get("i[:]"));
      assertTrue(i>=-32768 && i<=32767);
   }


}"
```

Apêndice C: Plano de testes - Teste de integração.

Devido a natureza dos testes de fumaça, foi criado um plano de testes para cada nova entrega e construção do sistema, no entanto, aqui encontra-se documentada apenas o plano de testes referentes a última construção, para efeitos práticos.

Descrição do caso de teste	**Procedimento para execução**	**Resultado esperado**	**Resultado encontrado**
Efetuar login com cadastro inválido	Inserir um usuário não cadastrado na tela de login e solicitar acesso. Login inválido: Logininvalido@login.com Senha inválida: %&#$%	Sistema deverá apresentar uma mensagem de "Usuário ou senha inválidos", não permitindo o login.	Sistema apresentou uma mensagem de "Usuário ou senha inválidos" e não autorizou o acesso.
Tentar cadastro sem preencher todos os campos.	Solicitar cadastro preenchendo apenas os campos de nome e senha, deixando o campo de e-mail em branco.	Sistema deverá apresentar uma mensagem de erro com o campo que foi deixado em branco	Sistema apresentou uma mensagem de erro com o campo que foi deixado em branco, ressaltando novamente sua obrigatoriedade
Efetuar cadastro como professor	Solicitar cadastro preenchendo todos os campos com os seguintes dados: Tipo do cadastro: Professor; Nome: Alexander R Kutzke; Senha: TesteCadastro; E-mail: alexkutzke@gmail.c	Sistema deverá cadastrar o usuário com sucesso e então redirecionar para a tela de início do mesmo	Sistema cadastrou o usuário com sucesso e redirecionou para a tela de início.

	om		
Cadastrar questão sem Enunciado	Acessar a tela de cadastro de questões e preencher o título mas deixar o enunciado em branco, em seguida, clicar em salvar. Título: Questão Nova	Sistema não deverá cadastrar a questão e então exibir uma mensagem sobre a obrigatoriedade do campo que foi deixado em branco	O sistema não permite o cadastro da questão, exibindo a mensagem da obrigatoriedade do campo deixado em branco e não perdeu o preenchimento do título.
Criar um caso de teste em branco	Ao gerar um caso de teste novo, solicitar que as entradas sejam geradas pelo Ammit e que as saídas sejam digitadas manualmente.	Sistema deverá exibir mensagem de erro com os campos que são obrigatórios e não foram preenchidos	Sistema exibiu mensagem de erro com os campos obrigatórios deixados em branco.
Gerar casos de teste com sintaxe Ammit inválida	Ao selecionar para que as entradas sejam geradas automaticamente pelo Ammit, fornecer uma sintaxe inválida: Sintaxe inválida: Sintaxeinvalida	A caixa de texto com os exemplos deverá permanecer vazia.	A caixa com exemplos não gerou nenhuma entrada e só foi populada com um indicador de fim.
Selecionar saída inválida ao criar caso de teste.	Ao gerar um caso de teste novo, solicitar que as entradas sejam geradas pelo Ammit e que as saídas sejam digitadas manualmente.	A opção de envio manual de saídas não deve estar disponível para escolha caso o usuário selecione geração via Ammit durante as entradas.	A opção de envio manual de saídas foi ocultada quando o usuário selecionou a geração automatizada de casos de teste.
Durante a geração de casos de teste, enviar um arquivo que não compile.	Ao criar um caso de teste, selecionar o envio de arquivo para validação das respostas e então enviar um arquivo inválido (Com erros de sintaxe).	Sistema deverá acusar o erro no arquivo e não salvar o caso de teste até que um arquivo válido seja selecionado ou a forma manual seja	Sistema acusa erro de compilação no arquivo e impede que o professor cadastre o caso de teste até trocar o arquivo e tentar novamente.

		escolhida.	

Apêndice D: Plano de testes - Teste de validação

O plano de testes a seguir foi executado por um usuário externo ao projeto, para o qual foi fornecido o software em sua versão final, um breve tutorial sobre o que era esperado, o plano de testes e a documentação do levantamento de requisitos.

Foi idealizado pelos autores que o usuário selecionado tenha experiência com desenvolvimento de software e conhecimento sobre os assuntos abordados e estrutura da documentação (Requisitos funcionais, Plano de testes). Para este fim, o usuário voluntário foi o senhor Christopher Marcel de Dias Siqueira, formado em Jogos Digitais pela Universidade Positivo e Programador Jr na empresa Stefanini.

Plano de testes baseados nos requisitos funcionais

Descrição do caso de teste	Procedimento para execução	Resultado esperado	Resultado encontrado
(Professor)Cadastrar uma questão nova	Acessar o menu lateral “Questões” e selecionar a opção para cadastro. Preencher o Título e enunciado da questão e então clicar em salvar. Titulo: Teste Externo Enunciado: Criação de questão para a documentação de teste externo do web service Ammit.	A questão será criada e o usuário será direcionado para a tela de gerência de casos de teste para a mesma.	A questão foi criada e então ocorreu o direcionamento para a tela de gerencia de casos de teste.
(Professor)Criar um novo caso de teste	Após a criação de uma questão, clicar em “Criar novo caso de teste”, então preencher todos os campos e clicar em salvar. Título: Caso para questão externa.	Sistema compila o código enviado na saída e então cria o caso de teste com sucesso, tornando-o parte da correção quando um aluno submeter uma resposta para a pergunta em	Sistema compilou o código e então criou o caso de teste como parte da correção da questão criada

	Conteúdo: Caso de teste 01. Entrada: Sintaxe Ammit i[1:100, 10] Saída: Envio de arquivo fornecido pelos autores.	questão	
(Professor)Remover um caso de teste	Através do menu de Gerência de questões, selecionar a gerência de casos de teste e então remover o caso de teste “Caso para questão externa”	Sistema deve exibir uma mensagem de confirmação para exclusão e quando confirmado deve remover o registro	Sistema exibiu mensagem para a confirmação da exclusão e quando confirmado removeu o registro
(Professor)Editar uma questão	Através do menu de Gerência de questões, pesquisar a questão “Teste Externo Questão” e então selecionar a opção de edição da questão, e então renomeá-la para “Teste Externo” . Em seguida, clicar em salvar.	O sistema deve salvar a alteração e exibir uma mensagem de que a alteração foi concluída com sucesso. Quando a questão for acessada novamente ela deve possuir o novo título.	O sistema alterou a questão assim que foi clicado no botão salvar e em um novo acesso a alteração se encontrava salva com sucesso.
(Professor)Excluir a questão pesquisada	Através do menu de Gerência de questões, pesquisar a questão “Teste Externo” e então selecionar a opção de remover a questão. Confirmar a remoção quando solicitado.	O sistema deverá exibir uma mensagem para confirmação da exclusão e em caso positivo deverá excluir a questão e então exibir uma mensagem de “Questão removida com sucesso”	O sistema só excluiu a questão com confirmação, como desejado, e assim que terminou a exclusão exibiu uma mensagem informando.
(Aluno) Pesquisar uma questão	Através do menu “Questões para responder” pesquisar a questão “Teste externo”.	O sistema deve atualizar a lista de questões com as correspondências da pesquisa	A pesquisa funcionou corretamente, voltando às questões buscadas
(Aluno)Enviar resposta para a	Clicar na opção responder ao lado	Sistema deve enviar a resposta e então	Sistema preenche as listas de forma

questão	da pergunta pesquisada e então selecionar o arquivo com a resposta que deverá ser enviado, na sequência, clicar em salvar.	logo em seguida mostrar ao aluno o resultado da questão e duas listas: Uma de acertos e uma de erros.	correta, separando acertos e erros e mostrando como foi e como era o esperado para cada um deles.

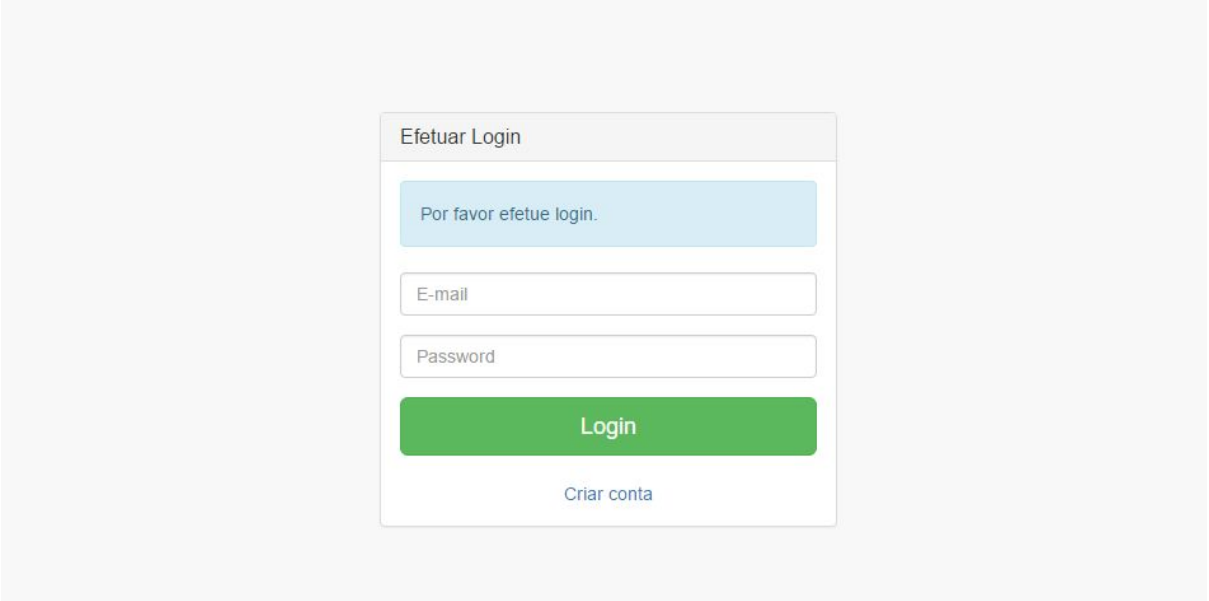
APÊNDICE E: Especificações de caso de uso

UC01 Entrar no sistema

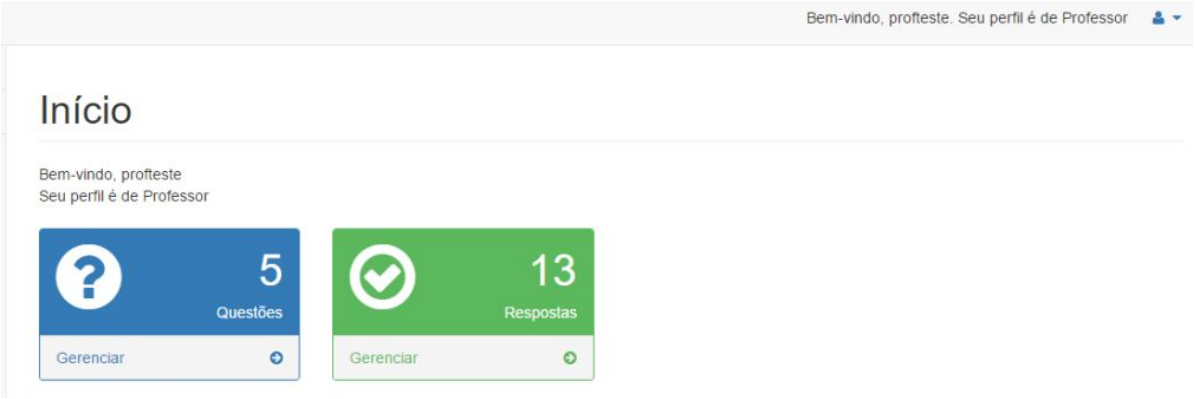
Descrição:

Este caso de uso serve para fazer login no sistema

Data View:



DV1: Página de login



DV2: Inicio professor

DV3: Inicio Aluno

Pré-Condições:

Não há

Pós-Condições:

Após o fim normal deste caso de uso, o sistema deverá:

1. Ter autenticado o cliente no sistema
2. Ter apresentado a tela inicial correspondente ao usuário logado

Ator Primário:

Usuário

Fluxo de Eventos Principal:

1. O sistema exibe a tela (DV1)
2. O usuário digita seu e-mail e senha e clica em Login (A1)
3. O sistema busca o usuário e senha (E1)
4. O sistema compara a senha apresentada com a senha do usuário (E1)
5. O sistema inicia a página inicial de acordo com o tipo de usuário (DV2)(DV3)
6. O use case atual é encerrado

Fluxos Alternativos

A1. O usuário clique em “Criar Conta”

1. O use case UC Cadastrar novo usuário é iniciado
2. O use case atual é encerrado

Fluxos de Exceção

E1. Nome de usuário não existe ou senha incorreta

1. O use case é reiniciado
2. O sistema exibe a mensagem “Nome de usuário e/ou senha incorretos” acima do campo de login

Regras de Negócio.

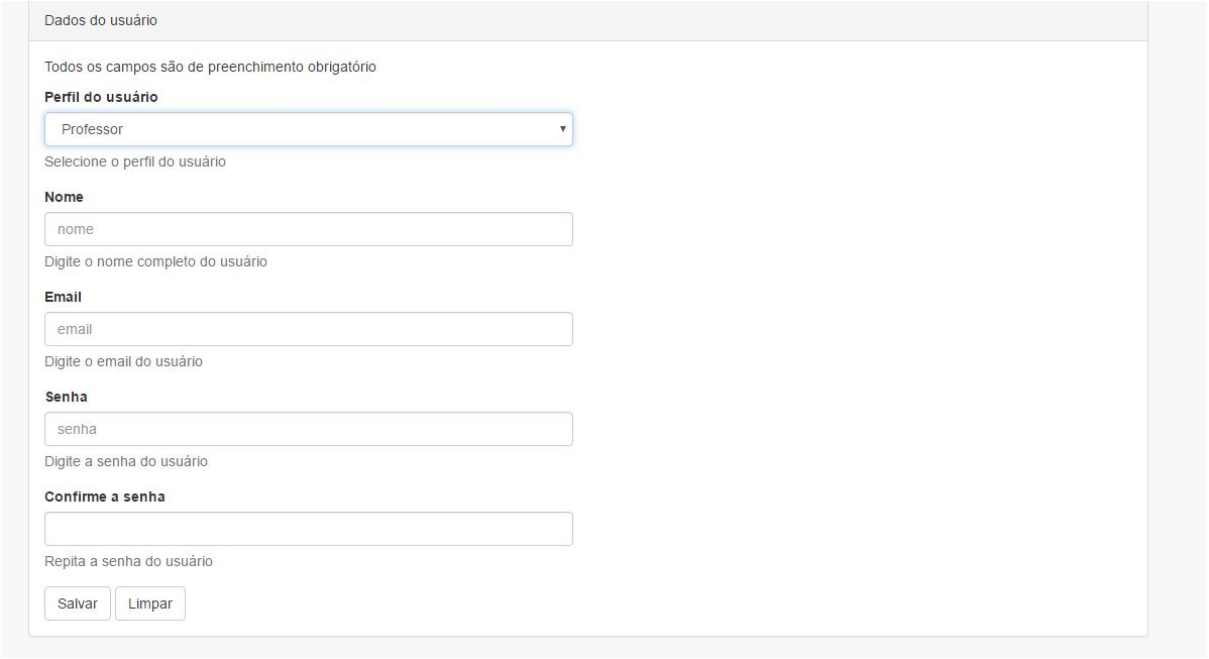
Não há

UC02 Cadastrar novo usuário

Descrição

Este caso de uso serve para criar um novo perfil no sistema

Data View



DV1: Página de cadastro

Pré-Condições

O Caso de uso deve ser iniciado através do UC Login

Pós-Condições

Após o fim normal deste caso de uso, o sistema deverá:

1. Ter enviado as informações do novo usuário para o banco de dados
2. Ter autenticado o usuário com o usuário recém cadastrado
3. Ter apresentado a tela inicial correspondente ao usuário logado

Ator Primário

Usuário

Fluxo de Eventos Principal

1. O sistema exibe a tela (DV1)
2. O sistema carrega os campos necessários para a criação de um novo cadastro
3. O sistema carrega a combobox de perfis com as opções disponíveis
4. O usuário seleciona o perfil
5. O usuário preenche todos os campos (E1)
6. O usuário clica em salvar (A1)
7. O sistema valida todos os campos (E2)
8. O sistema envia todas as informações do usuário para o banco de dados
9. O sistema autentica o usuário criado no sistema.
10. O sistema inicia a página inicial de acordo com o tipo de usuário (DV2)(DV3)
11. O use case atual é encerrado

Fluxos Alternativos

A1. clique em “Limpar”

1. O sistema apaga todas as informações de todos os campos
2. O sistema volta o UC para o passo 2.

Fluxos de Exceção

E1. O usuário não preenche todos os campos.

O sistema não salva as informações

O sistema exibe uma mensagem de erro com os campos faltosos

E2. Um ou mais dados inseridos são inválidos

O sistema não salva as informações

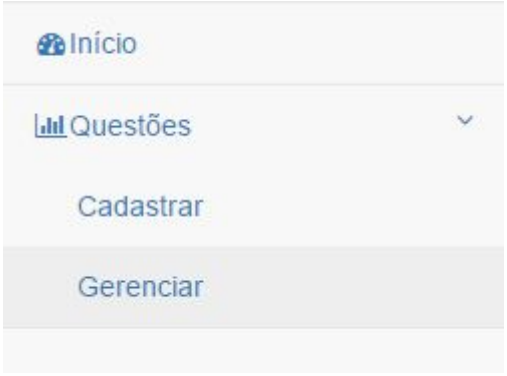
O sistema exibe uma mensagem de erro com os campos inválidos.

UC03 Gerenciar questão

Descrição

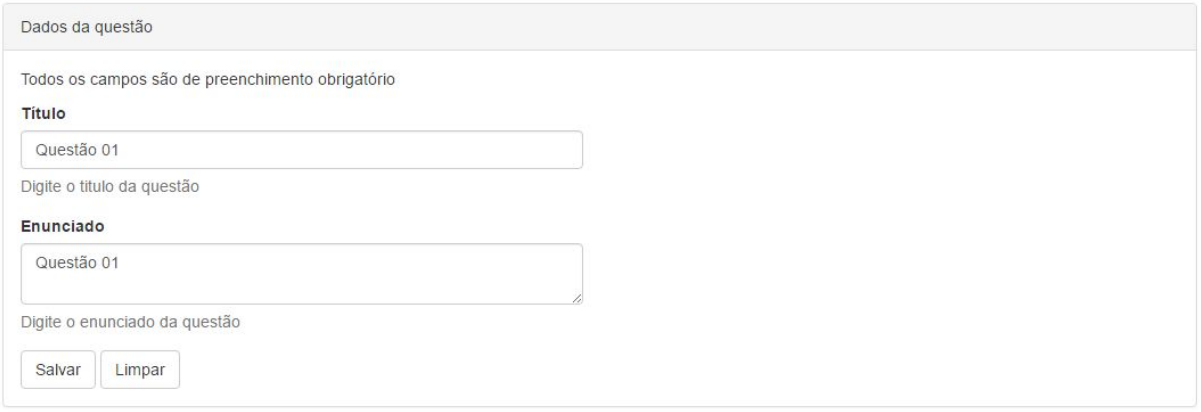
Este caso de uso é usado para descrever quando o professor busca uma questão no sistema e a altera de alguma forma.

DataView



DV1: Menu gerenciar

DV2: Lista de questões



DV2: Alterar uma questão

DV3: Excluir uma questão

Pré-condições:
O usuário autenticado deve ser do tipo professor
Pós-condições:
Não há.

Ator primário:
Professor
Fluxo de Eventos Principal.

1. O professor acessa o menu gerenciar pelo menu lateral (DV1)
2. O Sistema carrega um número estipulado de resultados com as questões cadastradas pelo mesmo (Número padrão: 10)(DV2)
3. O professor pesquisa a questão desejada
4. Na coluna de opções o professor seleciona para editar a questão selecionada. (A1)(A2)
5. O sistema carrega a tela com mais detalhes da questão para que a mesma seja alterada.(DV3)
6. O professor altera os campos desejados (E1)
7. O professor clica em salvar (A3)
8. O sistema salva as alterações
9. O sistema informa que salvou as alterações
10. O caso de uso é encerrado

Fluxos Alternativos:

A1: O professor seleciona a opção de excluir questão.

1. O sistema exibe uma mensagem de confirmação (DV4)
2. O professor confirma a remoção
3. O sistema exclui a questão referida.
4. O UC retorna para o passo 2.

A2: O professor seleciona a opção de gerenciar casos de teste.

1. O sistema inicia o UC Gerenciar Caso de teste para a questão selecionada.
2. O caso de uso é encerrado.

A3: O professor clica em limpar:

1. O sistema desfaz todas as alterações do professor na questão
2. O sistema volta o caso de uso para o passo 5

Fluxos de Exceção
E1: O professor deixa em branco algum dos campos.

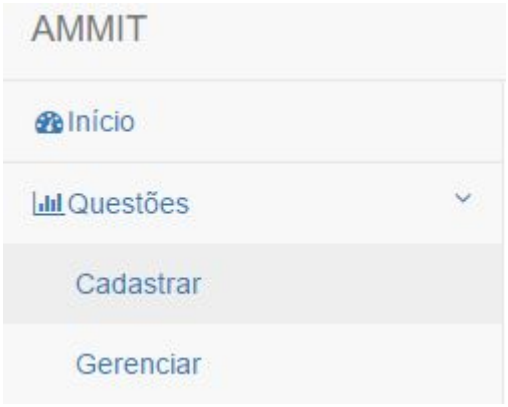
1. O sistema alerta para os campos em branco

2. O sistema não salva as alterações.

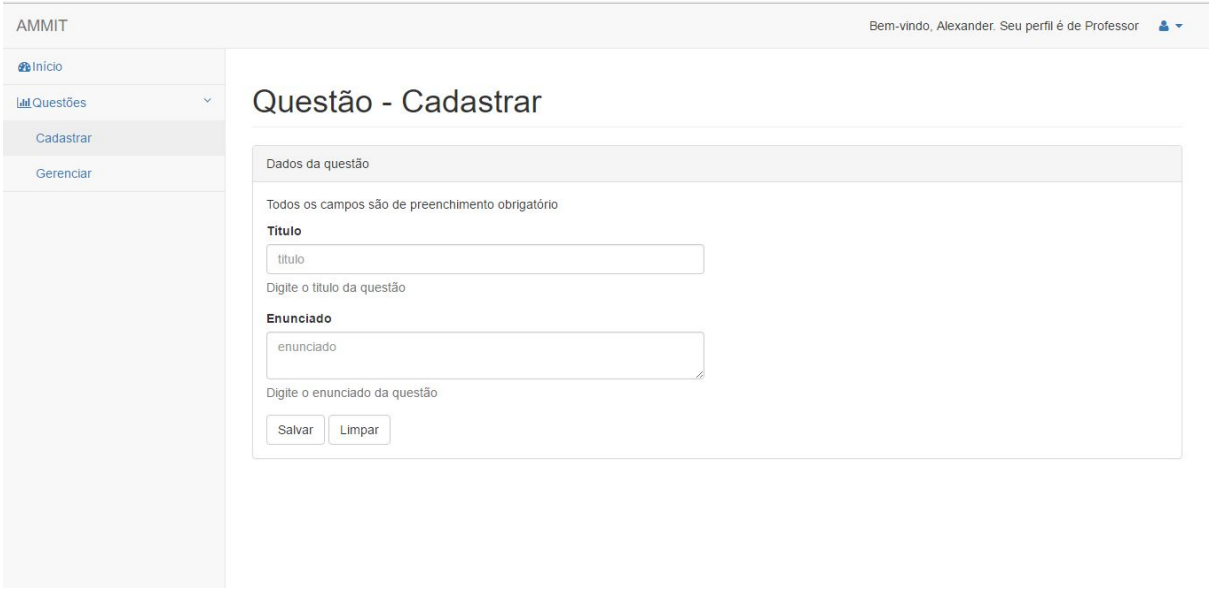
Regras de negócio
Não há

UC04 - Cadastrar questão.

Descrição
Este caso de uso descreve o processo para criação de uma nova questão.
DataView:



DV1: Menu Cadastrar



DV2:Nova questão

Pré-condições:
O usuário autenticado deve ser do tipo professor

Pós-condições:
Não há.
Ator primário:
Professor
Fluxo de Eventos Principal.

1. O professor acessa o menu cadastrar pelo menu lateral (DV1)
2. O Sistema carrega a tela para o cadastro de uma nova questão (DV2)
3. O professor preenche os campos com as informações desejadas (E1)
4. O professor clica em salvar (A1)
5. O sistema salva a questão no banco de dados
6. O sistema inicia o UC - Gerenciar casos de teste
7. O caso de uso é encerrado.

Fluxos Alternativos:
A1: O professor clica em limpar:

1. O sistema desfaz todas as alterações do professor na questão
2. O sistema volta o caso de uso para o passo 5

Fluxos de Exceção
E1: O professor deixa em branco algum dos campos.

1. O sistema alerta para os campos em branco
2. O sistema não salva as alterações.

Regras de negócio:
Não há

UC05 - Gerenciar casos de teste

Descrição

Este caso de uso descreve o processo de gerenciamento de casos de teste e suas possíveis operações.

DataView:

DV1: Lista de casos de teste
DV2: Detalhes de caso de teste
DV3: Confirmação de exclusão
DV4: Ajuda sobre a sintaxe.

Pré-condições:

1. Este UC só pode ser chamado através do UC Gerenciar questões ou do UC Cadastrar questão
2. O usuário autenticado deve ser do tipo professor

Pós-condições.

Não há

Ator Primário:
Professor
Fluxo de Eventos Principal:

1. O sistema carrega a tela com a lista de casos de teste (DV1).
2. O sistema preenche a listagem de casos de teste com os casos de teste já cadastrados para a questão, se existirem.
3. O Professor clica em “Criar novo caso de teste”.(A1)(A2)
4. O sistema carrega a tela para a criação de um novo caso de teste (DV2) com os campos em branco.
5. O Professor preenche o título e conteúdo do caso de teste.
6. O Professor seleciona “Utilize o Ammit para gerar entradas” como método de entrada(A3)
7. O Professor preenche o campo da entrada com uma expressão da linguagem Ammit.
8. O Professor seleciona quantas entradas deverão ser testadas.
9. O Professor clica em gerar (A4)(E1)
10. O Professor seleciona o arquivo a ser enviado para validar as saídas (E2)
11. O Professor clica em salvar (A5)(E3)
12. O Sistema salva as informações no banco de dados
13. O Caso de uso volta para o primeiro passo.

Fluxos de eventos alternativos.

A1: O Professor seleciona para editar um caso de teste

1. O sistema carrega a tela com detalhes de um caso de teste (DV2) com os campos preenchidos com as informações salvas no banco de dados.
2. O professor altera os campos conforme desejado.
3. O professor clica em salvar(A5)(E3)
4. O sistema salva as informações no banco de dados
5. O caso de uso volta para o primeiro passo.

A2: O Professor seleciona para remover um caso de teste.

1. O sistema exibe uma mensagem pedindo a confirmação sobre a remoção.(DV3)
2. O professor confirma a remoção
3. O sistema exclui o caso de teste do banco de dados e da lista.
4. O caso de uso volta para o primeiro passo.

A3: O professor seleciona “Entradas manuais”

1. O Professor preenche o campo da entrada com cada entrada que ele deseja testar manualmente
2. O Professor seleciona o arquivo a ser enviado para validar as saídas (E2)(A6)
3. O Professor clica em salvar (A5)(E3)
4. O Sistema salva as informações no banco de dados
5. O Caso de uso volta para o primeiro passo.

A4: O professor clica em “Aprenda mais sobre o Ammit”

1. O sistema abre uma janela flutuante com as informações sobre a sintaxe.(DV4)
2. O professor fecha a janela.
3. O caso de uso retoma sua execução no fluxo principal.

A5: O professor clica em limpar:

1. O sistema desfaz todas as alterações do professor no caso de teste

A6: O Professor seleciona “Insira manualmente suas saídas”

1. O professor define manualmente as saídas que espera para cada entrada definida manualmente
2. O professor clica em salvar (A5)
3. O sistema salva as informações no banco de dados.
4. O caso de uso volta para o primeiro passo.

Fluxos de Exceção

E1: Sintaxe Ammit inválida ao clicar em gerar.

1. O sistema não gera nenhuma entrada e a caixa de prévia permanece vazia.

E2: O arquivo enviado pelo professor possui erros de sintaxe e não compila.

1. O Sistema exibe uma mensagem sobre o erro de compilação e não permite o salvamento do caso de teste até que outro arquivo seja enviado.

E3: O professor deixa em branco algum dos campos.

1. O sistema alerta para os campos em branco
2. O sistema não salva as alterações.

Regra de negócio.

1. Quando as entradas são geradas através do Ammit, as saídas não podem ser definidas manualmente.

UC06 - Enviar resposta

Descrição
Caso de uso para descrever o processo de envio de respostas por parte do aluno.

DataView

DV1: Menu questões para responder.
DV2: Lista questões
DV3: Detalhes da questão

Pré-condições.
O perfil deve ser autenticado como aluno

Pós-condições.
Iniciar o UC exibir resposta

Ator primário
Aluno

Fluxo de evento principal:

1. O aluno acessa a listagem de questões através do menu lateral (DV1)
2. O sistema carrega a tela (DV2)
3. O sistema carrega a lista com as questões ainda pendentes para resposta para aquele aluno.
4. O aluno seleciona a questão a qual deseja enviar uma resposta e clica no botão correspondente
5. O sistema carrega a tela com as informações sobre a questão selecionada (DV3)
6. O Aluno seleciona o arquivo para ser enviado como resposta (E1)
7. O aluno clica em salvar (A1)
8. O sistema inicia o UC Exibir resposta
9. Fim do caso de uso.

Fluxo alternativo

A1: O aluno clica em limpar

1. O sistema remove o arquivo selecionado
2. O sistema retorna o caso de uso para o passo 5

Fluxo de exceção

E1: O arquivo enviado pelo aluno possui erros de sintaxe e não compila.

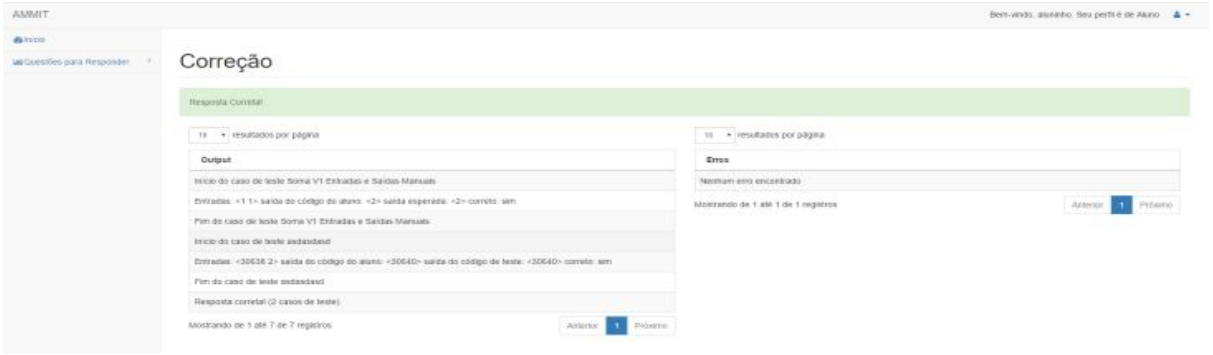
1. O Sistema exibe uma mensagem sobre o erro de compilação e não permite o salvamento da resposta até que outro arquivo seja enviado.

Regra de negócio
Não há

UC07 - Exibir resposta

Descrição
Caso de uso para a descrição do processo de exibição de correção, sendo a resposta correta ou não.

DataView



DV1: Correção de exercício.

Pré-condições

1. Só pode ser executado se chamado pelo UC Enviar resposta

Pós-condições
Não há

Ator primário.
Usuário

Fluxo de eventos principal:

1. Com a chamada do UC, o sistema carrega a tela (DV1)
2. O Sistema preenche as listas de acertos e erros com os resultados da correção
3. O Sistema exibe uma mensagem com o parecer final sobre a questão.
4. Fim do caso de uso.

Fluxos Alternativos:
Não há

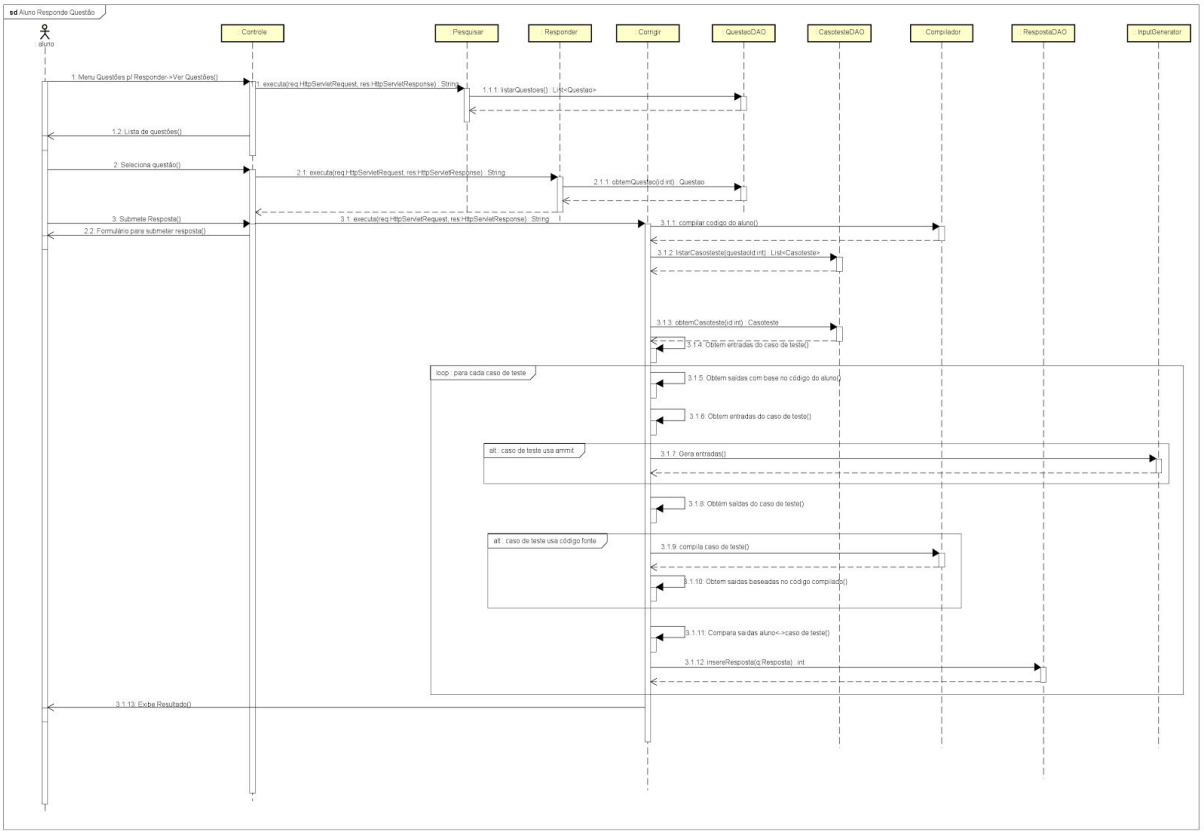
Fluxos de Exceção
Não há

Regra de Negócio
Não há

APÊNDICE F: Diagramas de Sequência

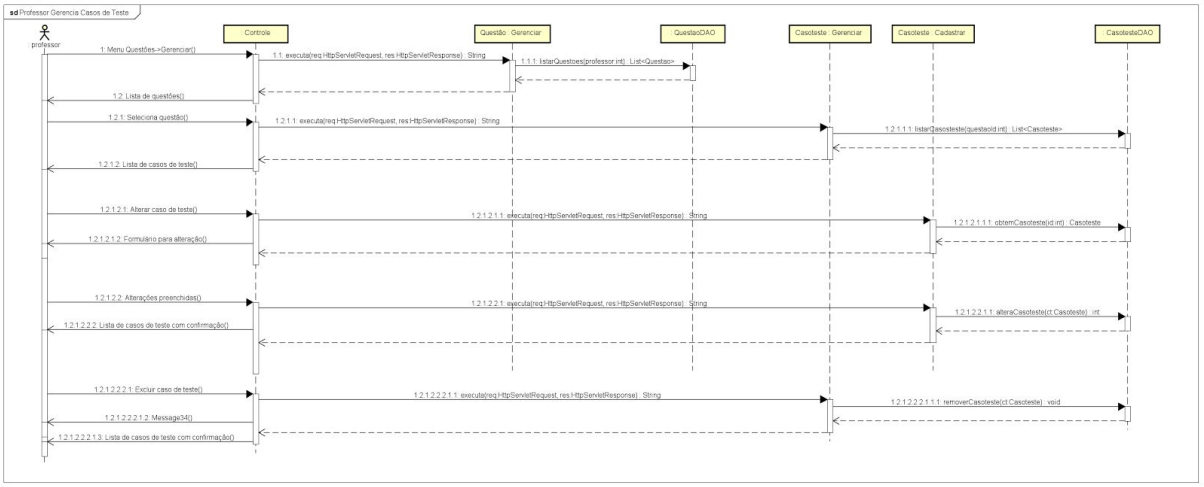
F1. Aluno Responde Questão

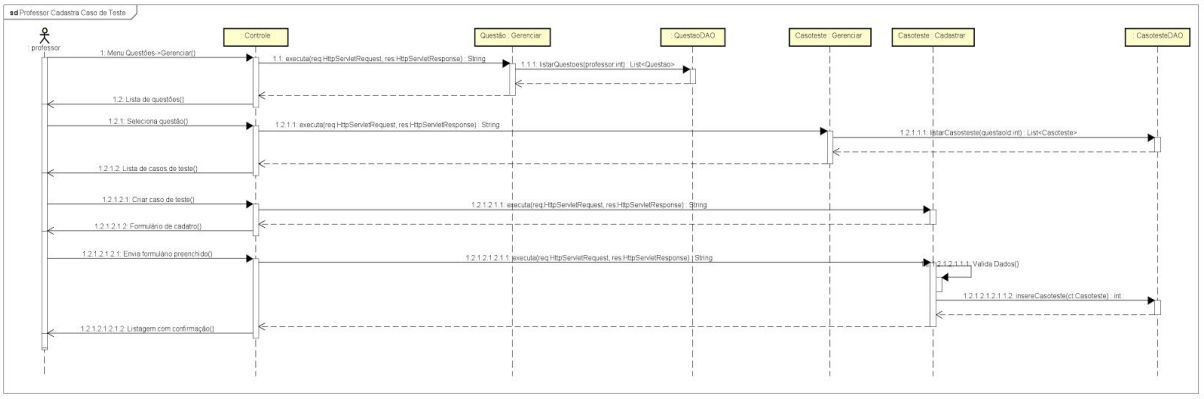
Este diagrama representa o fluxo de eventos que envolvem um aluno respondendo uma questão cadastrada no sistema



F2. Professor Gerencia/Cadastra Casos de Teste

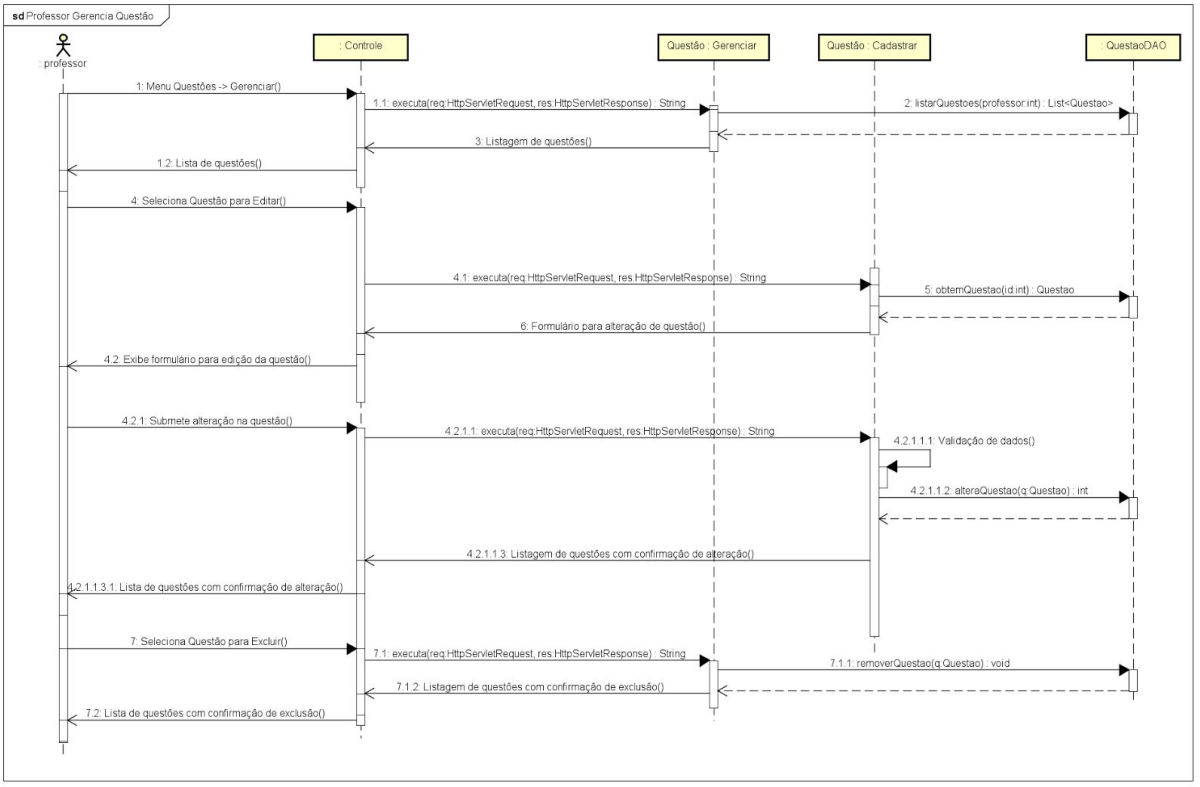
Estes diagramas representam o fluxo de eventos que envolvem um professor gerenciando os Casos de Teste de uma questão





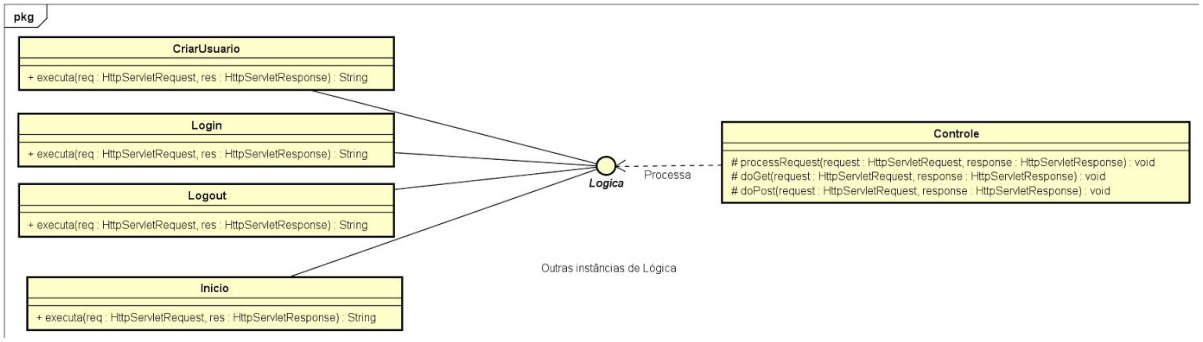
F3. Professor Gerencia Questão

Estes diagramas representam o fluxo de eventos que envolvem um professor gerenciando questões

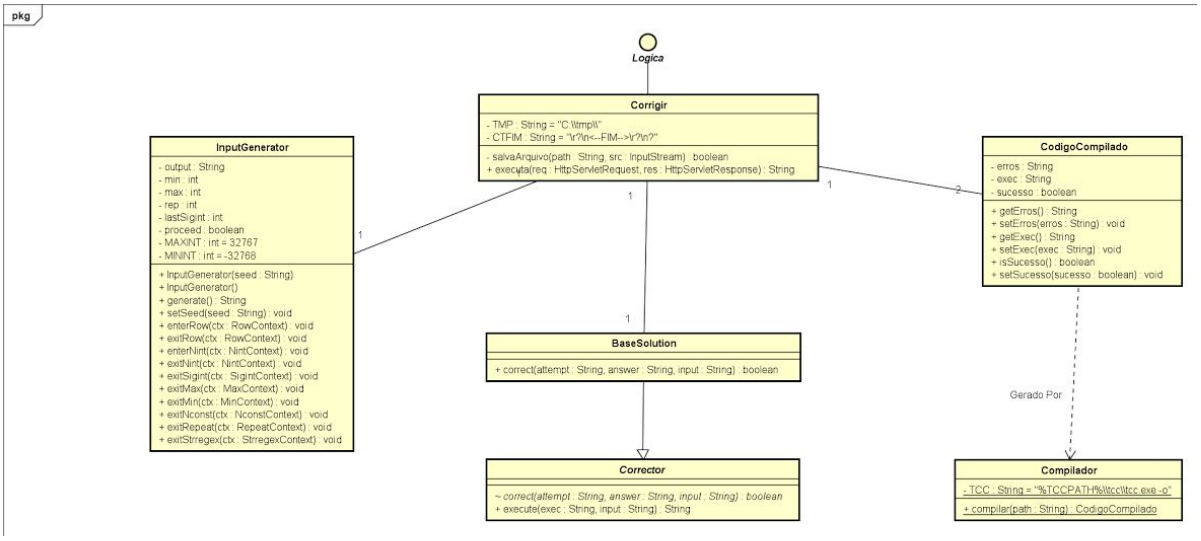


APÊNDICE G: Diagramas de Classe

G1. Classes de Controle de MVC



G2. Classes de correção de resposta



G3. Classes de entidades